

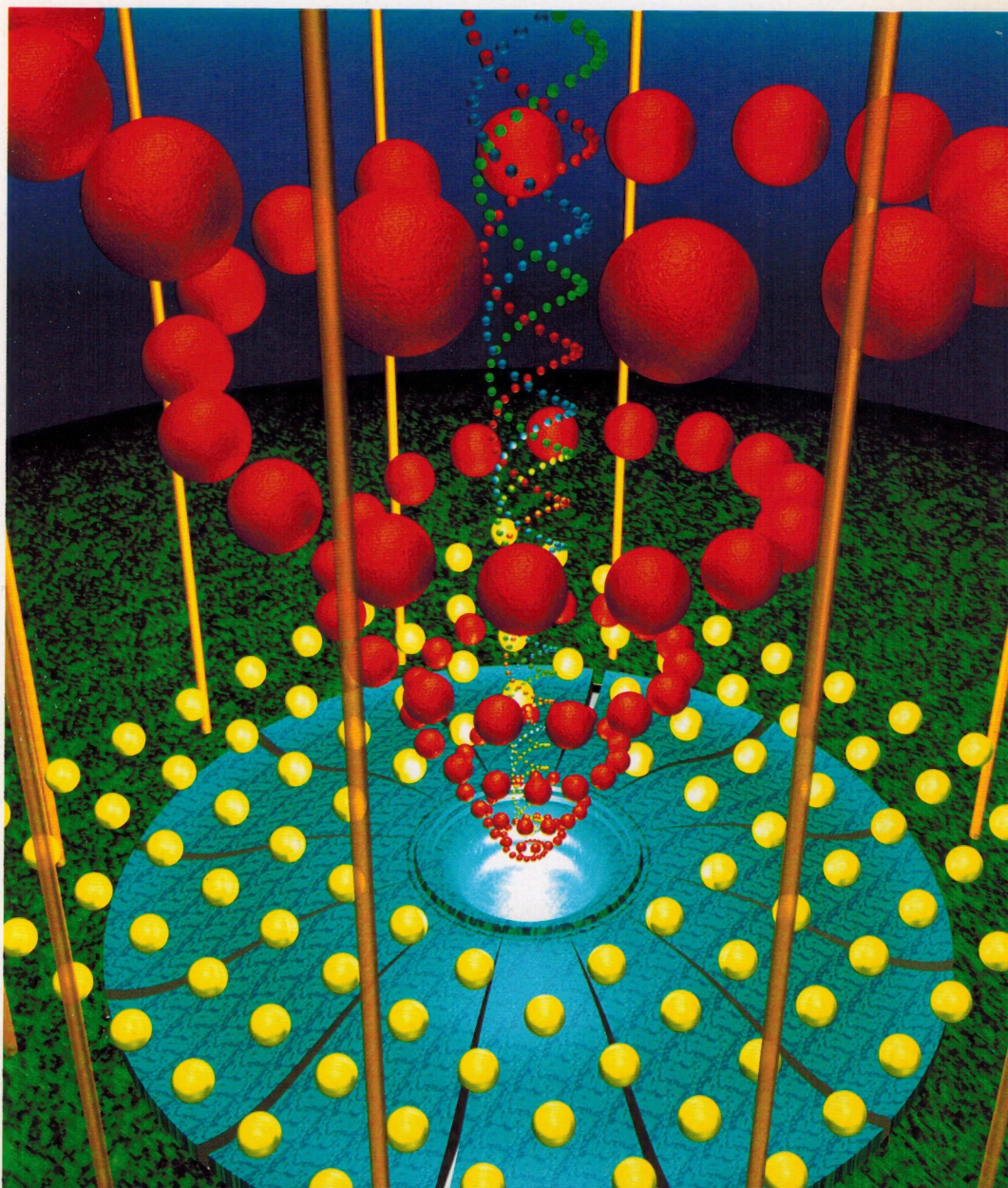
# トランジスタ技術

SPECIAL

No.49

## 特集 徹底解説 Z80マイコンのすべて

Z80CPUの概要から周辺LSIの活用法, ICEによるデバッグまで





エレクトロニクスの基礎と実用技術を濃縮したフィールド・ワーク・マガジン

# トランジスタ技術 **SPECIAL**

季刊●B5判●定価：①～③③定価1,570円 ③④～④⑤定価1,631円 ④⑥～④⑨定価1,723円 ⑤⑩～⑤⑦定価1,835円 ⑤⑧以降定価1,840円

①個別半導体素子 活用法のすべて 基礎からマスタするダイオード、トランジスタ、FETの実用回路技術	②③回路デザイナのためのPLD最新活用法 PLDのプログラミング法からPALライタの製作まで	④③Cによるマイコン制御プログラミング 86系ペリフェラルを中心とした
③PC9801と拡張インターフェースのすべて 16ビット・パソコンを使いこなすためのハード&ソフト	②⑦ハードディスクとSCSI活用技術のすべて 本格活用のためのハード&ソフトのすべてを詳解	④④フィルタの設計と使い方 アナログ回路のキーポイントを探る
④C-MOS標準ロジックIC活用マニュアル 実験で作る4000B/4500B/74HCファミリ	②⑧最新・電源回路設計技術のすべて 3端子レギュレータから共振型スイッチング電源まで	④⑤PC98シリーズのハードとソフト 386&486マシンを使いこなす!
⑤画像処理回路技術のすべて カメラとビデオ回路、パソコンと隔さる	②⑨マイコン独習Z80完全マニュアル 手作りの原点から実用ソフトの作成まで	④⑥アナログ機能ICとその使い方 民生用AV機器からマルチメディア分野で活躍する
⑥Z80ソフト&ハードのすべて 基礎からマクロ命令を使いこなすまでのノウハウを集成	③⑩ニュー・メディア時代のデータ通信技術 赤外線、無線通信技術からLAN、光ファイバを用いた高速通信技術まで	④⑦高周波システム&回路設計 通信新時代の回路技術とシステム設計
⑧データ通信技術のすべて シリアル・インターフェースの基礎からモデムの設計法まで	③①基礎からのビデオ信号処理技術 複合映像信号の理解からハイビジョン信号の捉え方まで	④⑧作れば解るCPU ロジックICで実現するZ80とキャスル・マシン
⑨パソコン周辺機器インターフェース詳解 セントロニクス/RS-232C/GPIB/SCSIを理解するために	③②実用電子回路設計マニュアル アナログ回路の設計例を中心に実用回路を詳述	④⑨徹底解説 Z80マイコンのすべて Z80CPUの概要から周辺LSIの活用法、ICEのデバッグまで
⑩IBM PC&80286のすべて 世界の標準パソコンとマルチタスクの基礎を理解する	③③オプト・デバイス応用回路の設計・製作 光素子を使いこなすための製作ドキュメント	⑤⑩フレッシャーズのための電子工学講座 電磁気学の基礎から電子回路の設計、製作までをやさしく解説
⑪フロッピーディスク・インターフェースのすべて 需要の急増するFDDシステムの基礎から応用まで	③④つくるICエレクトロニクス 機能ICを使って実用機器を作ろう (在庫僅少)	⑤①データ通信技術基礎講座 RS232Cの徹底理解からローカル通信の実用技術まで
⑬シミュレータによる電子回路理論入門 コンピュータを使ったアナログ回路設計の手法を理解するために	③⑤C言語による回路シミュレータの製作 Quick Cでのプログラミングとフィルタ回路の解析	⑤②ビデオ信号処理の徹底研究 映像信号の基礎から高画質化のためのデジタル信号処理の方法まで
⑭技術者のためのCプログラミング入門 MS-C、Quick C、Turbo Cによるソフトウェア設計のすべて	③⑥基礎からの電子回路設計ノート トランジスタ回路の設計からビデオ画像の編集まで	⑤③パソコンによる計測・制御入門 研究室や実験室に必要なデータ収集のノウハウを基礎から解説
⑮アナログ回路技術の基礎と応用 計測回路技術のグレードアップをめざして (在庫僅少)	③⑦実用電子回路設計マニュアルⅡ 豊富な回路設計例から最適設計を学ぼう	⑤④実践パワー・エレクトロニクス入門 パワーMOS FETとIGBTの使い方をやさしく解説
⑯A-D/D-A変換回路技術のすべて アナログとデジタルを結ぶ最新回路設計ノウハウ	③⑧Z80システム設計完全マニュアル 周辺I/Oボードの設計とマイコン・システムの開発	⑤⑤作ってわかる電子回路製作入門 やさしい電子工作からパソコンを使ったシステム開発まで
⑰OPアンプによる回路設計入門 アナログ回路の誤動作とトラブルの原因を解く	③⑨A-Dコンバータの選び方・使い方のすべて アナログ信号をデジタル処理するための基礎技術	⑤⑥電子回路シミュレータ活用マニュアル アナログ回路解析だけでなくデジタル回路解析も追加された
⑲PC9801計測インターフェースのすべて オリジナル拡張ボードでパソコンを実践活用しよう	④⑩電子回路部品の活用ノウハウ 機器の性能と信頼性を支える受動部品の使い方	⑤⑦最新・スイッチング電源技術のすべて 効率とノイズを重点的に解説したソフト・スイッチングの指南書
⑳アナログ回路シミュレータ活用術 ゲーム感覚の回路設計を体験しよう	④①実験で学ぶOPアンプのすべて 汎用OPアンプから高性能OPアンプまで	⑤⑧基本・C-MOS標準ロジックIC活用マスタ 低電圧動作とドライブ能力の向上をはかった
㉒デジタル回路ノイズ対策技術のすべて TTL/CMOS/ECLの活用法と誤動作/トラブルへの処方	④②高速デジタル回路の測定とトラブル解析 ハイスピード・デジタル信号を高周波と捉えられる	⑤⑨新世代Z80CPUで学ぶマイコン入門 RISCライクなZ80互換プロセッサKC80を詳解する

**CQ出版社** ☎170 東京都豊島区巣鴨1-14-2 販売部 ☎03-5395-2141 振替 00100-7-10665

定価は1997年4月1日現在の消費税5%を含んだ表示です



# トランジスタ技術

CONTENTS

## SPECIAL No.49

### 特集 徹底解説 Z80 マイコンのすべて

Z80CPUの概要から周辺LSIの活用法、ICEによるデバッグまで

私たちの生活を支える縁の下の力持ち

第0章	マイコンとはなんだろう〈野口智樹〉	2
-----	-------------------	---

Z80 とはどんなCPUだろう

第1章	Z80 CPUの概要〈額田忠之〉	6
-----	------------------	---

Z80 を自由自在に動かすための

第2章	Z80 のレジスタと命令〈野口智樹〉	17
-----	--------------------	----

アクセス・タイミングの計算からバンク切り替えまで

第3章	メモリ周辺回路の設計〈村田浩義/北川信孝〉	26
-----	-----------------------	----

外部装置との情報のやり取りの窓口

第4章	I/O周辺回路の設計〈村田浩義/北川信孝〉	44
-----	-----------------------	----

電源ONからマイコンを動作させるまでの回路

第5章	クロック & リセット周辺回路の設計〈野口智樹〉	61
-----	--------------------------	----

割り込みの受け付けから割り込み処理の開始まで

第6章	Z80 の割り込みシステムの動作〈末木 豊/野口智樹〉	69
-----	-----------------------------	----

Appendix	Z80 の割り込み動作の詳細〈磯沼 薫〉	90
----------	----------------------	----

スイッチ入力やLEDの点灯をコントロールする

第7章	Z80 PIOの使い方〈野口智樹〉	92
-----	-------------------	----

時間を計ったり、クロック数を数える

第8章	Z80 CTCの使い方〈野口智樹〉	104
-----	-------------------	-----

シリアル通信の初期化から送受信プログラムまで

第9章	Z80 SIOの使い方〈高見 豊〉	111
-----	-------------------	-----

東芝TMPZ84C0xx/ザイログZ84Cxx/川崎製鉄KL5C8012の使い方

第10章	周辺組み込み & Z80 互換高速CPUの活用〈高見 豊/菅原尚伸/菊地恭徳/北 孝〉	130
------	---	-----

アセンブラによるソフト開発からICEによるデバッグまで

第11章	Z80 マイコン・システム開発手順〈野口智樹/藤丘勝信〉	150
------	------------------------------	-----

■ 頒布ディスクの内容と申し込み書		167
-------------------	--	-----



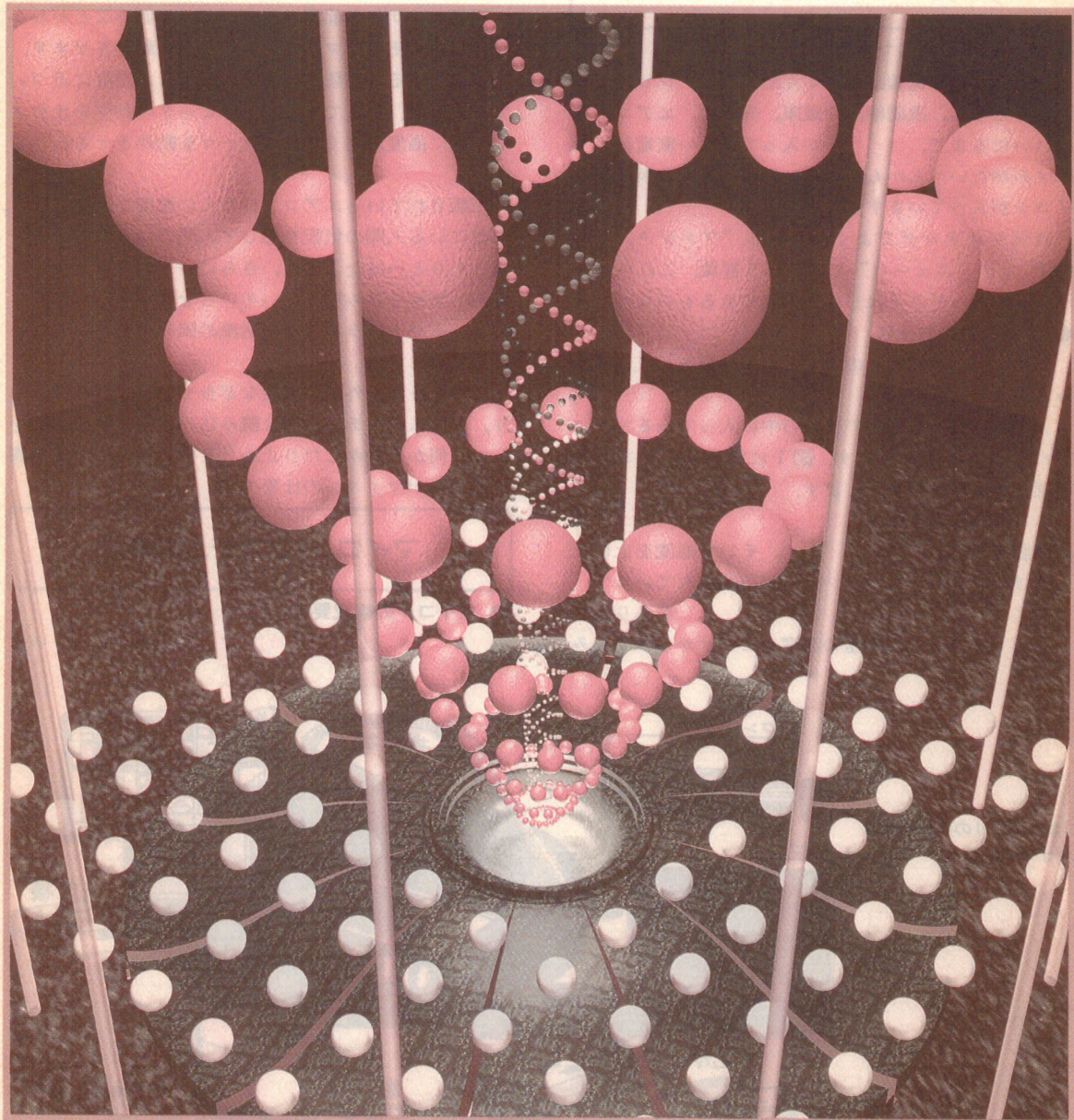




# 特集 徹底解説 Z80 マイコンのすべて

Z80 CPUの概要から周辺LSIの活用法、ICEによるデバッグまで

本誌はZ80 マイコンの概要からメモリやI/Oの接続方法、PIOやCTC、SIOの周辺LSIの使い方、そして周辺LSIを組み込んだものやZ80互換高速CPUについて、その活用方法を徹底解説します。さらにZ80を使ったマイコン・システムの開発の流れ、アセンブラの使い方からICEを使ったデバッグの方法も解説します。





# 第0章

私たちの生活を支える縁の下の力持ち

## マイコンとはなんだろう

野口智樹

### 家電製品の中に潜むマイコン

#### ● どこにでもあるマイコン

身の回りの家電製品をみてください。テレビやビデオはもちろん、洗濯機、冷蔵庫、掃除機、エアコンなどにいたるまで、**マイコン**が入っていない家電製品を探すのがたいへんかもしれないほど、多くの製品にマイコンが入っています。

宣伝やカタログでよくみかける“ファジー”だの“AI”だの“ニューロ”だのという言葉、これらはマイコンがあつてこそはじめて実現できるものなのです。

#### ● マイコンのお仕事

マイコンは家電製品の中でどんな仕事をしているかみてみましょう。

ここでは例としてエアコンの中に入っているマイコンについて考えてみます。図1に簡単なエアコンの内部ブロックの例を示します。

エアコンには部屋の**温度を測定する温度センサ**、人間が設定する**温度設定スイッチ**、そして温度を上げたり下げたりする**ヒータやクーラ**が内蔵されています。そして**これらをコントロールするマイコン**があります。

まずマイコンは、部屋の温度をセンサから取り込み

ます。そして温度設定スイッチの指示値を読み込んで、測定した温度と比較します。

温度センサから部屋の温度がだいぶ低いことがわかりました。そこでマイコンはヒータを動作させます。しばらくすると部屋が暖まり、指定した温度に近づいてきました。そこでマイコンはヒータの動作を止めます。逆に温度が高くなればクーラを動作させるわけです。

市販されているエアコンはさらに湿度を測定したり、部屋にいる人間の気配を感じて人間のいる方向へ風を送ったりといったように、さらに多くのセンサをつけ外部の情報を読み取り、いろいろなデータから判断してヒータやクーラの温度や送風の風向きをコントロールしています。

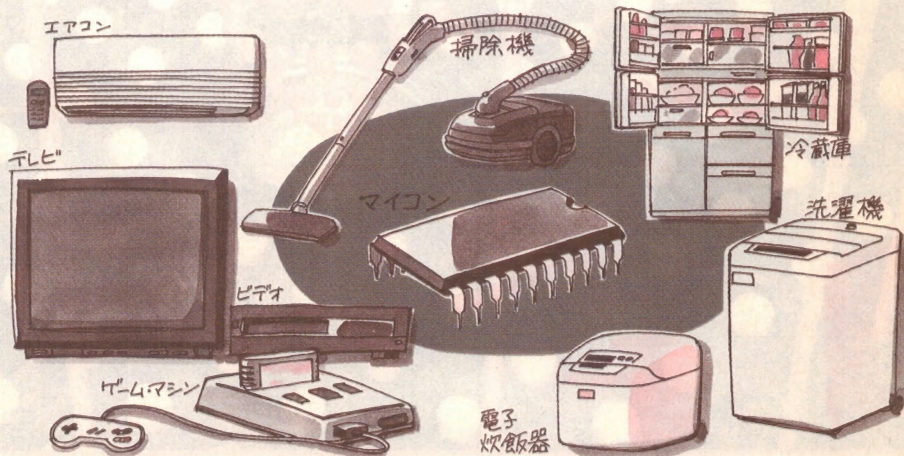
このように、外部の温度などをセンサで読み取ったり、設定スイッチの状態を読み込み、何らかの判断をして、何かを動かしたり止めたりするという作業を繰り返すのがマイコンの仕事です。

### マイコン・システムの構成

#### ● マイコンの構成要素

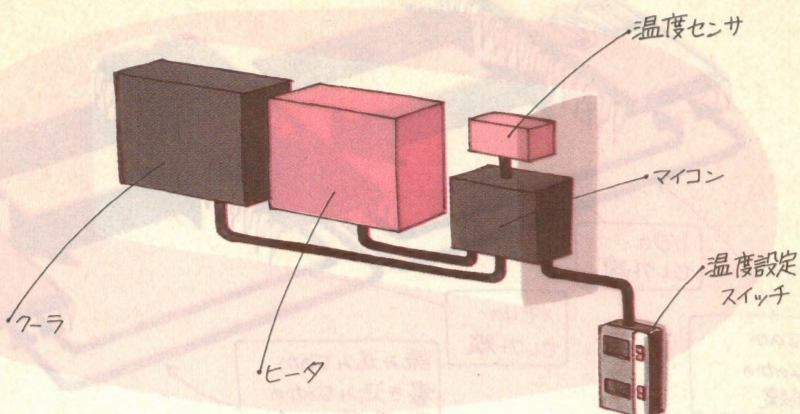
**マイコンは三つの基本的な要素から成り立っています。**

〈身のまわりの家電製品はマイコンだらけ〉





〈図1〉 エアコンの内部ブロック図



す。まずいちばん重要なのが、**計算をしたり判断をしたりするCPU**(中央演算装置)と呼ばれる部分です。そしてそのCPUを動作させるための**プログラムを記憶したり、入力されたデータを保存したりするためのメモリ**、もう一つは**外部から信号を読み込んだり、外部機器をコントロールするための入出力(I/O)部分**です(図2)。

これらの三つのうちどれか一つが欠けても、マイコン・システムは構成できません。また三つの要素があるからといって、かならず3個の部品から成るというわけでもありません。ワンチップ・マイコンとよばれるものは、これら三つの要素が一つのICの中に組み込まれています。

#### ● アドレス/データ・バス、制御線

これらCPU、メモリ、I/Oを接続するのが**アドレス・バス/データ・バス**というデータ線の束です(図3)。

アドレス・バスとは、その名のとおりにメモリやI/Oの番地を指定するための線で、データ・バスとは、アドレス・バスで決めたメモリやI/Oの番地にデータを書き込んだり読み込んだりするときの、**データの通り道**です。

またこれとは別に、メモリなのかI/Oなのかを指定する線、データの読み込みか書き込みかの制御線など、何本かの**制御線**があります。

#### ● マイコンの基本動作

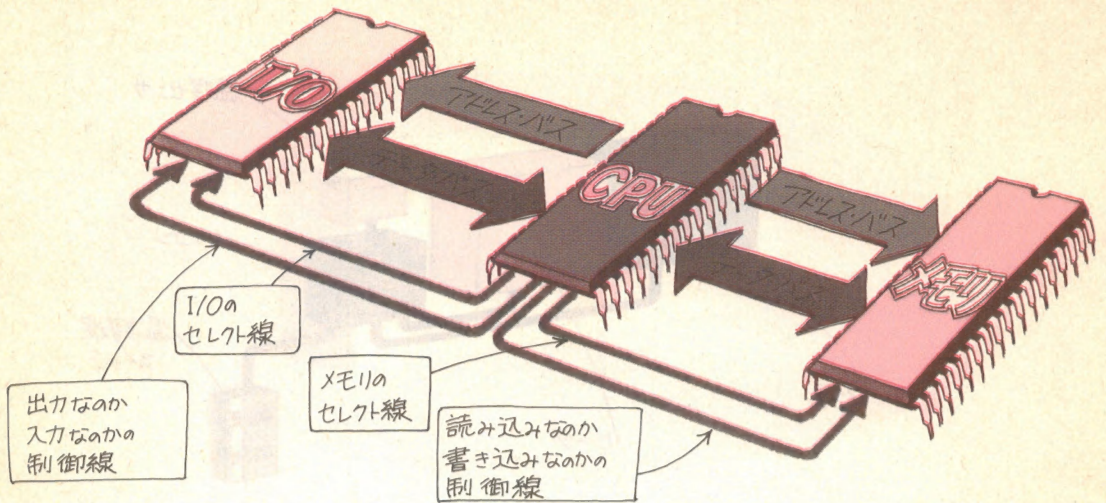
先ほど説明したエアコンの例に戻しましょう。温度

〈図2〉 マイコンの3要素





〈図3〉 アドレス・バスとデータ・バス



センサからの信号や、人間が設定した温度スイッチの情報の入力、またヒータやクーラのコントロール出力は、マイコンの I/O に接続されています。

“温度センサの出力を読み込み”や“ヒータを ON しろ”というプログラムはメモリに記憶しておきます。

電源 ON でエアコンが動き出すと、まずマイコンはアドレス 0 の命令をメモリから読み込みます。

この命令は“温度センサの出力を読み込み”でした。そこで今度はアドレス・バスに温度センサの I/O アドレスを出力し、センサからのデータを入力します。

そしてまた次の命令を読み込みます。“温度設定スイッチの指示値を読み込み”でした。今度は温度設定スイッチの I/O アドレスをアドレス・バスに出力し、設定スイッチの状態を入力します。

またまた次の命令を読み込み、温度センサからのデ

ータと設定スイッチからのデータとどちらが温度が高いかを判断します。

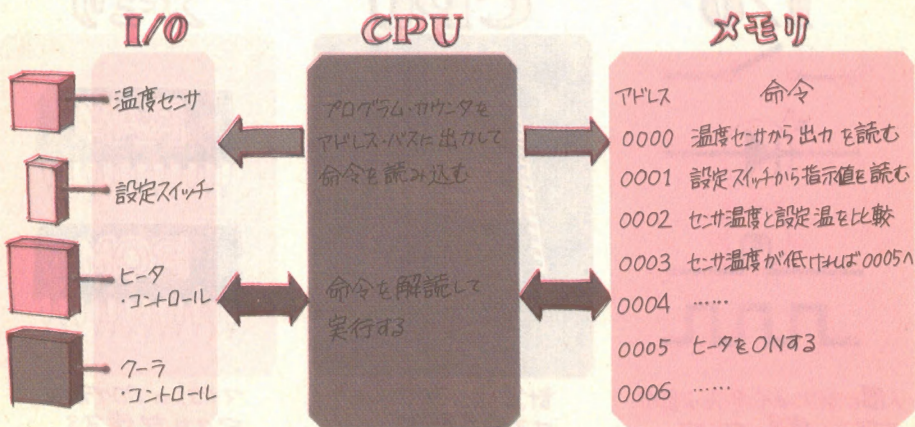
そして温度センサから読み込んだ部屋の温度が低ければアドレス 5 にジャンプします。

アドレス 5 の命令は、ヒータを ON しろという命令です。アドレス・バスにヒータの I/O アドレスを出力し、ヒータを ON するという意味の FFh というデータを出力します。これでヒータが ON されます (図 4)。

このようにマイコンは、命令を読み込んで、その命令に従って別のメモリ・アドレスのデータを読み込んだり、I/O アドレスにアクセスしてデータを入出力します。

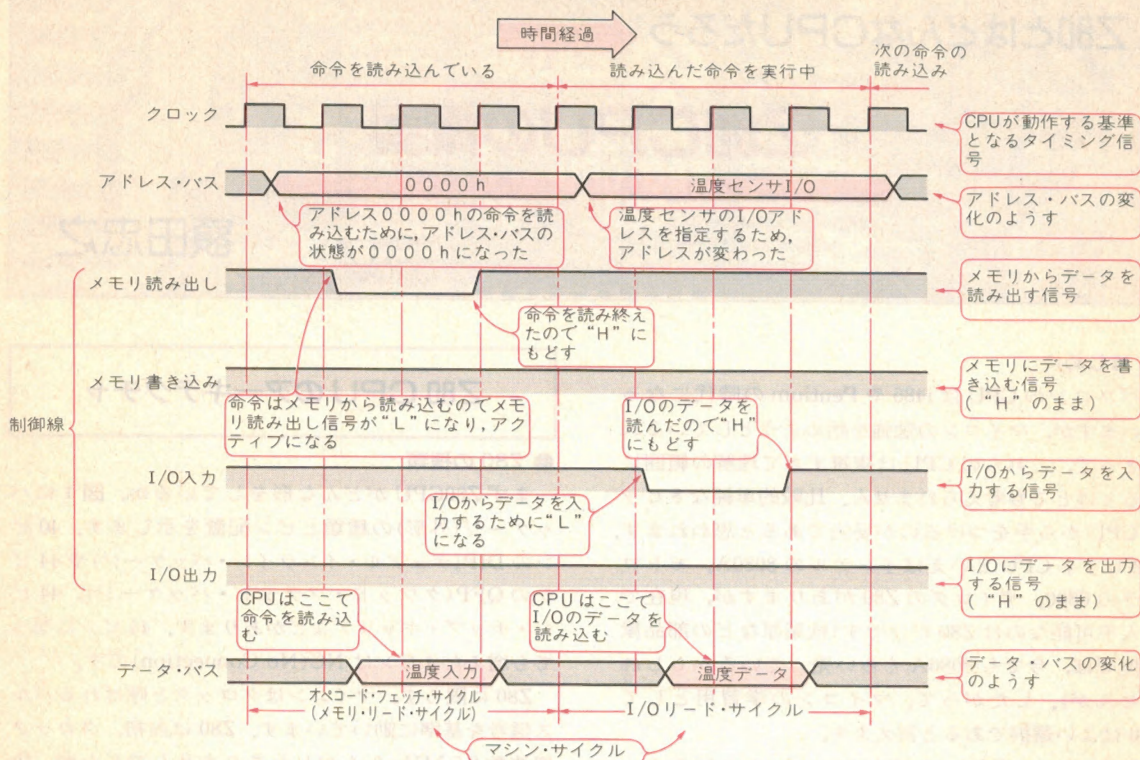
そしてふたたび次の命令を読み込んで動作を繰り返します。

〈図4〉 エアコンに内蔵されたマイコンの動作例





〈図5〉 マイコンの動作



## ● CPU 動作のまとめ

図5にCPUの細かな動作の移り変わりを示します。CPUが動作を繰り返すには、**クロック**と呼ばれる基準信号が必要です。人間でいえば心臓の鼓動のようなものでしょうか。クロック周波数が早ければ、それだけ高速なCPUといえます。

CPUが命令を読み込むには、まず読み込みたい**命令のアドレスをアドレス・バスに出力**します。命令はメモリから読み出すわけですから、次のクロックのタイミングで**メモリ読み出し信号**が出力(アクティブ)されます。そして次のクロックのタイミングで、CPUは命令を読み込みます。次のクロックではもう命令を読み込んだ後なので、メモリ読み出し信号を戻します。これがCPUが命令を読み込むときの動作です。この一連の動作を、命令のことを**オペコード**と呼ぶことから、**オペコード・フェッチ・サイクル**と呼びます。またメモリからデータを読み出しているともいえるので、**メモリ・リード・サイクル**とも呼びます。

さて、次にこの命令の実行です。読み込んだ命令は温度センサのI/Oから温度データを読み込む命令でした。

そこで、まず温度センサの**I/Oアドレスをアドレス・バスに出力**します。次のクロックでは**I/O読み出し信号**が出力され、次のクロックでCPUはデータを

読み込みます。読み込みが終わったので、次のクロックではI/O読み出し信号は元に戻ります。

これで温度センサからのデータの読み込みが終わりました。この動作もI/Oからデータを読み込んでいるので、**I/Oリード・サイクル**と呼びます。

またこれらのメモリ・リード・サイクルやI/Oリード・サイクルなど、CPUの各行程を**マシン・サイクル**と呼びます。

ここでアドレス・バスやデータ・バスをみてください。時間経過にしたがって出力されているデータが次々に変化しています。ある瞬間は命令だったり、ある瞬間はI/Oのデータだったりします。これを**時分割制御**と呼びます。

CPUにはいろいろな種類がありますが、世の中にあるほとんどのCPUは、だいたいこのようなタイミングで動作していると思って間違いありません。

## ● なぜZ80か

これを否定しようと、本書の存在意義がなくなってしまうのですが…

やはりZ80は8ビット・マイコンとして広く使われている、この一言につきると思います。

それでは次章から、Z80CPUの概要や動作について、詳しく解説していきましょう。

(トランジスタ技術1994年6月号に加筆、修正)



# 第1章

## Z80とはどんなCPUだろう

### Z80CPUの概要

額田忠之

#### ● はじめに

パソコンのCPUはi486やPentiumの時代になっていますが、マイコンの勉強を始めようとしている人にとって、これらのCPUは複雑すぎて理解の範囲にあるとはとても考えられません。比較的単純な8ビットCPUから手をつけるのが妥当であると思われます。

8ビットCPUといえばインテルの8080A、モトローラの6800、ザイログのZ80がありますが、現在でも入手可能なのはZ80だけです(秋葉原などの部品屋を捜せば、今でも8080Aくらい売っているかもしれませんが)。したがって、マイコンの学習用としてZ80はよい選択であると言えます。

ここでは、Z80とはどんなマイコンなのか、Z80CPUの概要について解説します。

なお余談ですが、日本ではZ80は「ゼット・はちじゅう」または「ゼット・はちまる」と呼ばれていますが、米国人はもちろん「ズィ・エイティ」と呼んでいます。

#### Z80 CPU のアーキテクチャ

#### ● Z80の種類

まずZ80CPUがどんな形をしているか、図1にパッケージ(外形)の種類とピン配置を示します。40ピンのDIP(デュアル・インライン・パッケージ)や44ピンのQFP(クワッド・フラット・パッケージ)、44ピン・チップ・キャリアなどがあります。44ピンになっても増えた4ピンはNC(No Connection)です。

Z80に限らず、マイコンはクロックと呼ばれるパルス信号を基準に動いています。Z80は当初、クロック周波数2.5MHzのものしかありませんでしたが、少しずつ改良されて現在では表1に示すように最高20MHz動作のものまであります。

#### ● Z80の各種信号線

図1に示した外観でわかるように、Z80には40本のいろいろな信号名のついたピンがあります。それぞれの名称や意味、動作を図2と表2に示し説明します。

どの信号線もそれぞれ重要な意味があるのですが、ここでは特に重要な信号線について説明します。

#### ▶ アドレス・バス

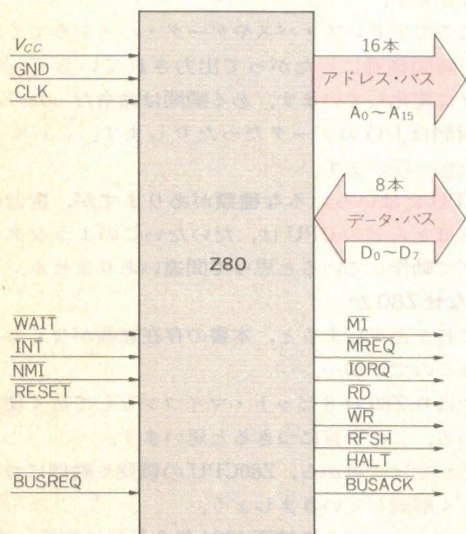
バスとは信号線の束と考えてください。ということは、アドレス・バスとは番地を示す信号線の束という意味になります。第0章で説明したように、マイコンは命令やデータをメモリから読み込みます。またセンサの信号をI/Oから読み込みます。このとき、どのメモリから読み込むのか、またはどのセンサからデータを読み込むかの番地を指定する必要があります。この指定のためにアドレス・バスが使われます。

Z80にはアドレス・バスが16本あり、それぞれA<sub>0</sub>～A<sub>15</sub>という名前がついていて、2<sup>16</sup>つまり0000h～0FFFFh(hは16進数を示す添え字)までの64Kバイトのメモリを管理することができます。

#### ▶ データ・バス

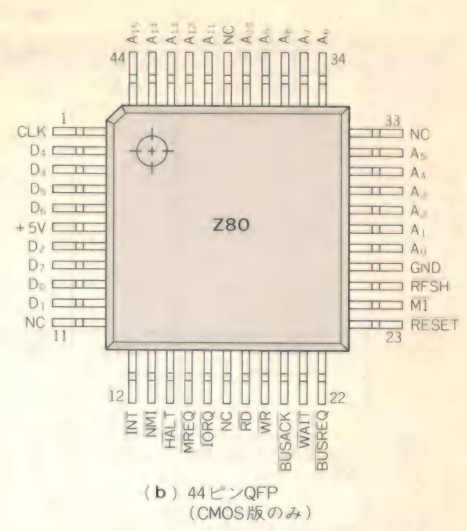
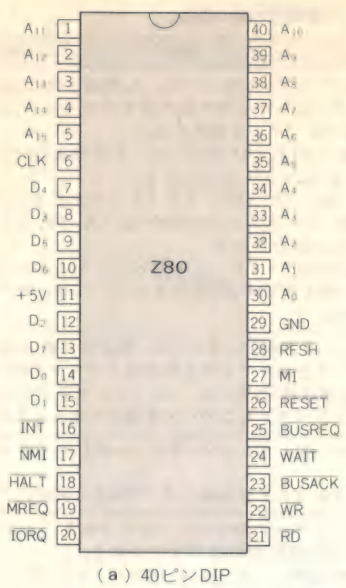
アドレス・バスが番地を示す信号の束なら、データ・バスはデータを示す信号の束となります。メモリ

〈図2〉 Z80CPUの信号線





<図 1>  
 Z80CPU の  
 パッケージ

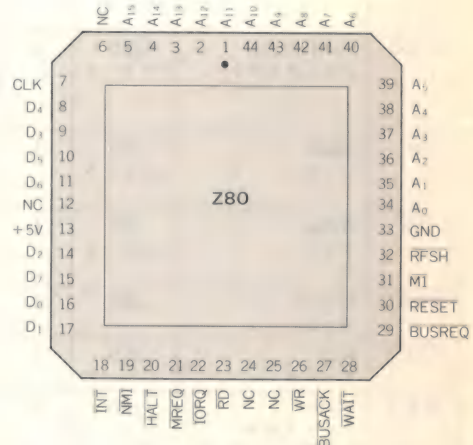


<表 1> Z80CPU の型名と最高動作速度

NMOS タイプ	最高動作周波数
Z0840004	4 MHz
Z0840006	6.17 MHz
Z0840008	8 MHz

CMOS タイプ	最高動作周波数
Z84C0006	6.17 MHz
Z84C0008	8 MHz
Z84C0010	10 MHz
Z84C0020	20 MHz



から読み込んだ命令や I/O からのデータは、このデータ・バスを通して CPU に読み込まれます。Z80 では  $D_0 \sim D_7$  の 8 本の線があります。

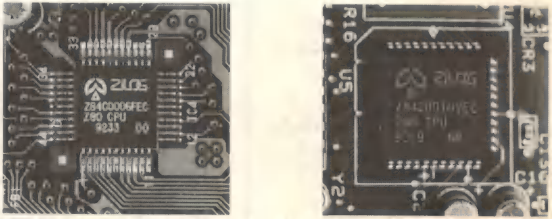
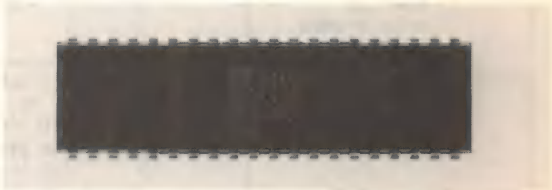
▶ **MREQ, IORQ**

マイコンには命令やデータを読み書きするためのメモリと、センサからの信号を読み込む I/O があります。これらの番地の指定にアドレス・バスが使われると説明しました。つまり番地指定にはメモリ、I/O にかかわらずアドレス・バスが兼用されるわけです。

そこでメモリを読みたいのか、I/O を読みたいのかという指定を別の信号線で指定する必要があります。そのための信号線がこの  $\overline{\text{MREQ}}$  と  $\overline{\text{IORQ}}$  です。メモリを読み書きしたいときは  $\overline{\text{MREQ}}$  が、I/O を読み書きしたいときは  $\overline{\text{IORQ}}$  が L レベルになります。

メモリか I/O かを区別していますが、ハードウェア的には  $\overline{\text{MREQ}}$  と  $\overline{\text{IORQ}}$  のどちらの信号がアクティブ(有効になる)かという違いだけです。

▶ **RD, WR**



<写真 1> Z80CPU の外観(上：DIP、左下：QFP、右下：チップ・キャリア)

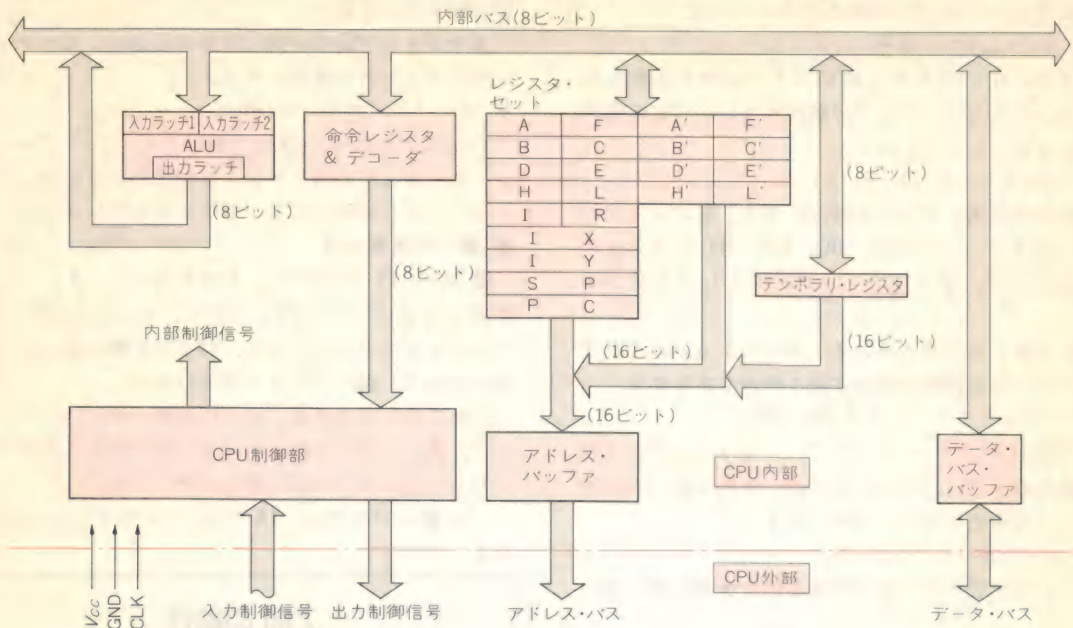


〈表 2〉 Z80CPU の各種信号線と機能

ピン名	信号線名	入/出力	有効極性	機 能
A <sub>0</sub> ~A <sub>15</sub>	Address Bus アドレス・バス	出力 3 ステート	H	<ul style="list-style-type: none"> <li>16ビットのアドレス・バス。A<sub>0</sub>が最下位(LSB)で、A<sub>15</sub>が最上位(MSB)。</li> <li>メモリに対しデータの読み書きをするときに、このバスにより 64 K バイト中の 1 バイトを選択する。</li> <li>IN A, (n) および OUT (n), A 命令実行時には、I/O ポート・アドレス、n が A<sub>0</sub>~A<sub>7</sub>に出力される。</li> <li>IN r, (C), OUT (C), r などレジスタ C を I/O アドレスとする入出力命令では、レジスタ C の内容が A<sub>0</sub>~A<sub>7</sub>に出力され、レジスタ B の内容が A<sub>8</sub>~A<sub>15</sub>に出力される。</li> <li>DRAM 用のリフレッシュ・アドレスは A<sub>0</sub>~A<sub>6</sub>に出力される。</li> <li>DMA 要求許可時(BUSACK="L")には、A<sub>0</sub>~A<sub>15</sub>はハイ・インビデンスになる。</li> </ul>
D <sub>0</sub> ~D <sub>7</sub>	Data Bus データ・バス	入出力 3 ステート	H	<ul style="list-style-type: none"> <li>8 ビットの双方向データ・バス。D<sub>0</sub>が最下位(LSB)で、D<sub>7</sub>が最上位(MSB)。</li> <li>このバスを通じてメモリや I/O とデータのやりとりをする。</li> <li>DMA 要求許可時には、ハイ・インビデンスになる。</li> <li>割り込みベクタもこのバスを通じて読み込まれる。</li> </ul>
$\overline{\text{MI}}$	Machine Cycle One M1 サイクル	出力	L	<ul style="list-style-type: none"> <li>オペコード・フェッチ・サイクル(M1 サイクル)を実行していることを示す。</li> <li>割り込みベクタ要求時には、<math>\overline{\text{IORQ}}</math> が同時に "L" になる。</li> </ul>
$\overline{\text{MREQ}}$	Memory Request メモリ・リクエスト	出力 3 ステート	L	<ul style="list-style-type: none"> <li>メモリとデータのやりとりをするときに、この信号は "L" になる。データの方向は RD または WR で決まる。</li> <li>DRAM 用のリフレッシュ・アドレスを出力するときには <math>\overline{\text{RFSH}}</math> 信号とともに "L" になる。</li> <li>DMA 要求許可時には、ハイ・インビデンスになる。</li> </ul>
$\overline{\text{IORQ}}$	I/O Request I/O リクエスト	出力 3 ステート	L	<ul style="list-style-type: none"> <li>I/O とデータのやりとりをするときに、この信号は "L" になる。データの方向は RD または WR で決まる。</li> <li>割り込みベクタ要求時には、<math>\overline{\text{MI}}</math> が同時に "L" になる。</li> <li>DMA 要求許可時には、ハイ・インビデンスになる。</li> </ul>
$\overline{\text{RD}}$	Read リード	出力 3 ステート	L	<ul style="list-style-type: none"> <li>メモリまたは I/O からデータを読み込むときに "L" になる。メモリと I/O の識別は <math>\overline{\text{MREQ}}</math> と <math>\overline{\text{IORQ}}</math> により行われる。</li> <li>DMA 要求許可時には、ハイ・インビデンスになる。</li> </ul>
$\overline{\text{WR}}$	Write ライト	出力 3 ステート	L	<ul style="list-style-type: none"> <li>メモリまたは I/O に対してデータを書き込むときに "L" になる。メモリと I/O の識別は <math>\overline{\text{MREQ}}</math> と <math>\overline{\text{IORQ}}</math> により行われる。</li> <li>DMA 要求許可時には、ハイ・インビデンスになる。</li> </ul>
$\overline{\text{RFSH}}$	Refresh リフレッシュ	出力	L	<ul style="list-style-type: none"> <li>DRAM 用のリフレッシュ・アドレスが A<sub>0</sub>~A<sub>6</sub>に出力されていることを示す。</li> </ul>
$\overline{\text{HALT}}$	Halt ホールド	出力	L	<ul style="list-style-type: none"> <li><math>\overline{\text{HALT}}</math> 命令が実行したことを示す。この後、割り込みまたは NMI 要求が発生するまで、CPU は NOP 命令の実行を繰り返す。</li> </ul>
$\overline{\text{WAIT}}$	Wadt ウェイト要求	入力	L	<ul style="list-style-type: none"> <li>CPU がメモリまたは I/O をアクセスするとき、この信号を "L" にすると、CPU はウェイト・サイクルを挿入し、アクセス時間を延長する。</li> </ul>
$\overline{\text{INT}}$	Interrupt Request 割り込み要求	入力	L	<ul style="list-style-type: none"> <li>この信号が "L" であると、CPU は割り込み受け付け禁止条件が解除されたときに、<math>\overline{\text{IORQ}}</math> と <math>\overline{\text{MI}}</math> を "L" にすると同時に、割り込みベクタをデータ・バスを通して読み込む。</li> </ul>
$\overline{\text{NMI}}$	Non Maskable Interrupt マスク不可能な 割り込み要求	入力	L	<ul style="list-style-type: none"> <li>この信号の立ち下がりエッジを検出すると、CPU はその時点で実行していた命令の終了時にもっとも優先度の高い割り込みを発生する。</li> </ul>
$\overline{\text{RESET}}$	Reset リセット	入力	L	<ul style="list-style-type: none"> <li>この信号が "L" であると(最低 3 クロック期間)、CPU はつぎのようにリセットされる。</li> <li>(1) プログラム・カウンタの値はゼロになる。</li> <li>(2) 割り込み受け付けが禁止される。</li> <li>(3) インタラプト・レジスタ (I) がゼロ・クリアされる。</li> <li>(4) リフレッシュ・レジスタ (R) がゼロ・クリアされる。</li> <li>(5) 割り込みモードがゼロ (IM0) になる。</li> </ul>
$\overline{\text{BUSREQ}}$	Bus Request バス・リクエスト	入力	L	<ul style="list-style-type: none"> <li>この信号が "L" であると、CPU はその時点に実行していたマシン・サイクル終了時に、アドレス・バス、データ・バスならびに制御信号をハイ・インビデンスにし、バスの制御権を外部回路(例えば DMA コントローラ)に譲る。</li> </ul>
$\overline{\text{BUSACK}}$	Bus Acknowledge バス・アクノリッジ	出力	L	<ul style="list-style-type: none"> <li><math>\overline{\text{BUSREQ}}</math> 信号が "L" であることを認識し、バスをハイ・インビデンスにした後、この信号を "L" にしてバスが解放されたことを外部に知らせる。</li> </ul>
CLK	Clock クロック	入力	—	<ul style="list-style-type: none"> <li>シングル・フェーズの MOS レベルのクロック。</li> </ul>
				最小                      最大 "L"電圧    -0.3 V                      0.45 V "H"電圧    V <sub>CC</sub> -0.6 V                      V <sub>CC</sub> +0.3 V



〈図3〉 Z80CPU の内部構成



以上の信号線で、メモリかI/Oか、そしてどのアドレスかは指定できます。残るはそのアドレスに対してデータを読み込むのか書き込むのかの信号線が必要です。それがこのRDとWRです。

#### ▶ RESET

これはリセット信号入力です。これをLレベルにするとCPUは初期化状態になり、Hレベルにすると、メモリのアドレス0000hから命令を読み込んで動き出します。

#### ▶ MI

これはCPUがメモリから命令を読み込んでいるときにLレベルになる信号線です。

#### ● Z80の内部構成

図3にZ80の内部構成図を示します。CPUをバス幅で分類する呼び方がありますが、Z80は図で示すように、内部バスが8ビットなので、8ビットCPUに分類されています。

ブロック図は複雑そうですが、とりあえず重要な部分を説明します。

#### ▶ ALU (Arithmetic Logical Unit)

日本語訳では中央演算装置で、何のことはない、計算をする部分です。計算といっても、足し算や引き算、論理演算程度しかできません。

実はZ80はかけ算やわり算ができません。これらの計算は、足し算や引き算を組み合わせで計算します。

#### ▶ 命令レジスタ&デコーダ

CPUがメモリから読み込んだ命令が、どんな命令であるかを解読する部分です。ここで命令を解読して、

CPU制御部に信号を送ります。

#### ▶ CPU制御部

命令デコーダで解読された命令によって、CPU内部の各部に制御信号を送る部分です。

▶ アドレス・バス・バッファ/データ・バス・バッファ  
さきほど説明したアドレス・バスやデータ・バスをコントロールするところです。

アドレス・バスの役割は番地を指定することですから、必ず出力方向になります。

データ・バスはデータを読み込んだり書き込んだりするので、双方向に制御できます。

#### ▶ レジスタ・セット

レジスタとは、メモリから読み出したデータや計算した結果を、CPUがいったん覚えておくところです。

Z80には26個のレジスタがあります。これらのレジスタの中には使い方が完全に決まっているレジスタもあります。また特に決まっていないものを汎用レジスタと呼びます。

#### ▶ テンポラリ・レジスタ

このレジスタがZ80内部に本当に存在するかは不明ですが、これがあると考えたほうが、CPU内部の動作説明が理解しやすくなります。

#### ● Z80の各種レジスタ

それでは、Z80にはどんなレジスタがあるのでしょうか。特に重要なレジスタについて説明します。

#### ▶ アキュムレータ(A)

Z80でいちばん重要なレジスタがこのアキュムレータであるAレジスタです。というのも、Z80の各種



演算は必ず A レジスタに対して行われ、その演算結果も必ず A レジスタに格納されるからです。

#### ▶ フラグ・レジスタ (F)

例えば、A レジスタと B レジスタの値を比較したときに、どちらのレジスタの値が大きいかなどの結果を覚えておくレジスタです。

#### ▶ レジスタ B, C, D, E, H, L

これらは汎用レジスタと呼ばれます。各レジスタのサイズは 8 ビットですが、BC, DE, HL レジスタの組み合わせで、16 ビット・レジスタとしても使えます。

また汎用レジスタとはいえ、命令によってはそれぞれのレジスタを特殊な用途に使う場合もあります。

#### ▶ インデックス・レジスタ (IX, IY)

これは 16 ビット・レジスタで、メモリからデータ列を読み込むときに、プログラムの便利な使い方ができるように考えられたレジスタです。

#### ▶ プログラム・カウンタ (PC)

次にどのアドレスの命令を読み込むかを記憶してい

るレジスタです。メモリから命令を読み込むたびに 1 ずつ増えていきます。

またジャンプ命令が実行されたときは、そのジャンプ先のアドレスの値が入ります。

#### ▶ スタック・ポインタ (SP)

レジスタの値を一時的に保存したり、サブルーチン・コールなどの戻りアドレスを保存するために使われる、メモリ領域のアドレスを示すカウンタです。

#### ● 裏レジスタとは

図 3 の CPU 内部ブロック図を見ると、A レジスタや B レジスタなどと同じ名前で、それぞれに“'”のついたレジスタがあります。ちょうど裏に隠れている感じなので、裏レジスタと呼ばれます。

これらのレジスタは、まさに表裏一体というイメージで、裏レジスタと表レジスタを切り替える命令で、どちらを表に出すか切り替えて使います。

この裏レジスタは、IX や IY レジスタにはありません。

## Z80 の動作

次は Z80 の動作についてみてみます。Z80CPU が命令を実行する場合、以下の五つの基本動作(マシン・サイクル)を組み合わせで行います。

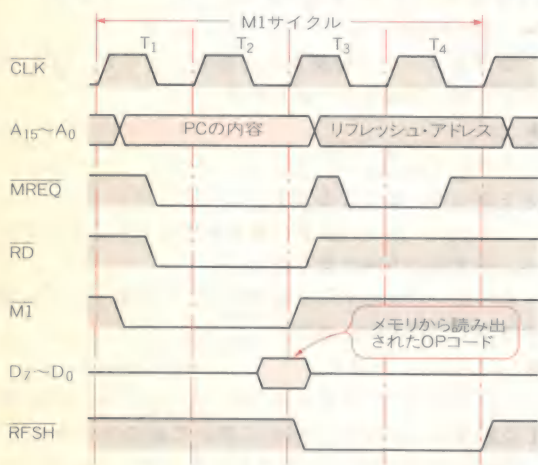
- (1) オペコード・フェッチ・サイクル(M1 サイクル)
- (2) メモリ・リード・サイクル
- (3) メモリ・ライト・サイクル
- (4) I/O リード・サイクル
- (5) I/O ライト・サイクル

では、まずこれらのマシン・サイクルについて説明し、その後で特定の命令がどのように実行されるかを説明します。

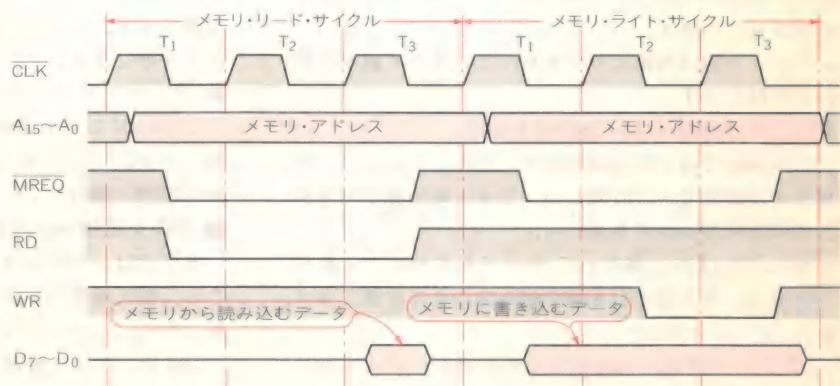
#### ● オペコード・フェッチ・サイクル

オペコード・フェッチとは、CPU が命令を読み込むことです。どの命令を実行する場合でも、最初のマシン・サイクルはオペコード・フェッチ・サイクルから始

〈図 4〉 オペコード・フェッチ・サイクル(M1 サイクル)



〈図 5〉  
メモリ・リード/ライト・  
サイクル





まります。その意味でこのマシン・サイクルは M1 サイクルとも呼ばれます。

このサイクルは命令のオペコードをメモリから読み取ります。図 4 にオペコード・フェッチ・サイクルのタイミングを示します。このマシン・サイクルは、4 クロック期間からなり、それぞれのクロック期間には  $T_1 \sim T_4$  という名前がつけられています。

まず  $T_1$  のときに、プログラム・カウンタの内容がアドレス・バス ( $A_0 \sim A_{15}$ ) に出力され、ついで  $\overline{M1}$ 、 $\overline{MREQ}$  と  $\overline{RD}$  が L レベルになります。

メモリから読み出されたデータ (命令のオペコード) は  $T_3$  の立ち上がりで CPU 内に取り込まれます。その後、アドレス・バス上にリフレッシュ・アドレスが出力され、 $\overline{RFSH}$  および  $\overline{MREQ}$  が L レベルになります (DRAM のリフレッシュ・サイクル)。

命令実行時には必ず一つの M1 サイクルが実行されますが、このサイクルが 2 回実行される命令もありま

す。命令の先頭バイトが DDh, EDh, FDh である命令はオペコードが 2 バイトになるので、M1 サイクルは 2 回実行されます。

### ● メモリ・リード/ライト・サイクル

このマシン・サイクルは、命令のオペコードに続くオペランド部分をメモリから読み取るとき、および命令が指示するメモリ・アドレスに対してデータの読み書きをする場合に実行されます。オペランドとは、たとえば “A レジスタと B レジスタを 〇〇 する” という命令のレジスタの指定にあたる部分です。

両サイクルは  $\overline{RD}$  が出力されるか、 $\overline{WR}$  が出力されるかが異なるだけなので、一緒に説明します。図 5 にメモリ・リード/ライト・サイクルのタイミングを示します。

メモリ・リード・サイクルでは、 $T_1$  のときにメモリ・アドレスが  $A_0 \sim A_{15}$  上にのせられた後、 $\overline{MREQ}$  および  $\overline{RD}$  が L レベルになります。メモリから読み出し

## Z80 七不思議 アーキテクチャ編

Z80 が誕生して約 20 年。その間の技術の進歩はめざましいものがあります。このコラムでは、今考えるとなぜ Z80 ではこうなっているのかといった点について、Q&A 方式で考えてみたいと思います。

**Q** : 8080 より Z80 はレジスタが強化されているが、A レジスタや HL レジスタをバンク切り替え方式にしたのはなぜか。EXX 命令で切り替えないとアクセスできないのでは面倒だし、一概にレジスタが増えたとはいえない気がする。

**A** : 内部回路を見ると、それぞれのレジスタを 2 組もたせ、フリップフロップを命令により切り替えるという文字どおりのバンク切り替えとなっており、もっとも簡単に増設する方法をとったことがうかがえます。

このデバイスの開発当時は、できるだけゲート数を少なくするということが絶対の時代でしたので、裏レジスタをサポートする命令もそのために組み込まれなかったと思われます。

**Q** : 演算命令がアキュムレータに集中していて命令の直交性が悪い。8080 から Z80 に強化するとき、2 バイト命令になったとしても、ADD B, C などのようにしていれば、命令も覚えやすくプログラムも作りやすくなったと思う。

**A** : Z80CPU の開発目標の一つは、8080 との命令セットの互換性ということにありました。また、8080 系の CPU は、専用レジスタのアーキテクチャをとっており、各レジスタの性格がはっきり決まっています。良かろうが悪かろうが互換性がないことには始まらないため、8080 のアーキテクチャをそのまま引きずってしまったのだと思います。

また当時では、8080 との互換性を保った状態で汎用レジスタ・アーキテクチャに変えることは、ゲート数の面からも難しかったと考えます。

**Q** : なぜ制御線が、 $\overline{MRD}$ ,  $\overline{MWR}$ ,  $\overline{IORD}$ ,  $\overline{IOWR}$  (メモリ・リード/ライト, I/O リード/ライト)ではなく、 $\overline{MREQ}$ ,  $\overline{IORQ}$ ,  $\overline{RD}$ ,  $\overline{WR}$  なのか。これではメモリをつなぐにも必ず OR ゲートが必要になってしまう。

**A** : これは、Z80 周辺 LSI の制約の関連からかとも思います。Z80 周辺 LSI では、CPU とのインターフェースに使用する信号線の数を節約するため、各信号に複数の線の組み合わせで各種トランザクションの区別を行うようになっています。 $\overline{IOWR}$  や  $\overline{IOWR}$  では、I/O リードまたは I/O ライトのときの状態しか示せませんし、両方がイネーブルにはなりません。これでは CPU の状態や割り込みベクタ要求などを周辺 LSI が知ることができません。〈信垣育司〉



れたデータは  $T_3$  の立ち下がりを読み込まれます。

またメモリ・ライト・サイクルでは、 $T_1$  のときにメモリ・アドレスが  $A_0 \sim A_{15}$  上にのせられ、 $\overline{MREQ}$  が L レベルになります。また、メモリに書き込まれるべきデータが  $D_0 \sim D_7$  上にのせられます。そして、 $T_2$  の立

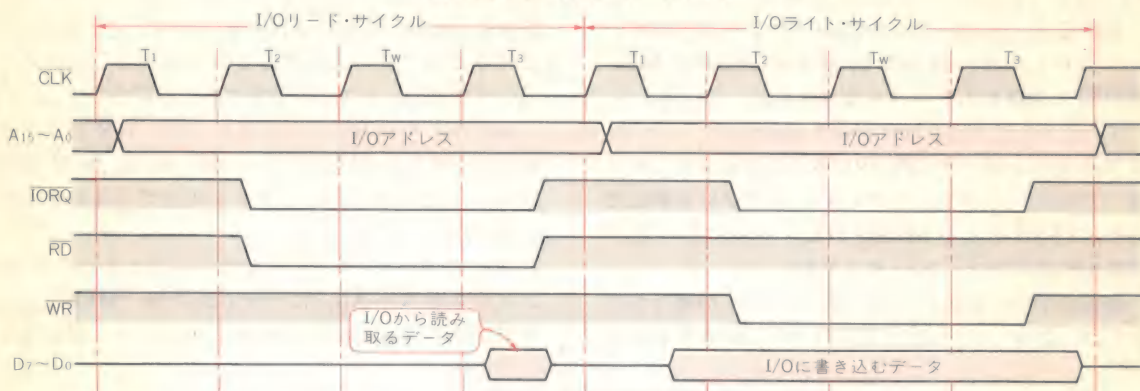
ち上がりで  $\overline{WR}$  が L レベルになります。

### ● I/O リード/ライト・サイクル

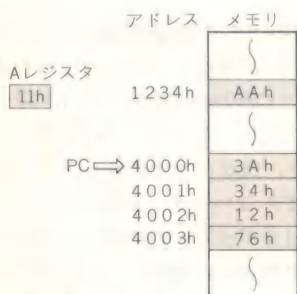
このマシン・サイクルもリード/ライト・サイクルは似ているので一緒に説明します。

このサイクルは **IN 命令または OUT 命令の実行時**

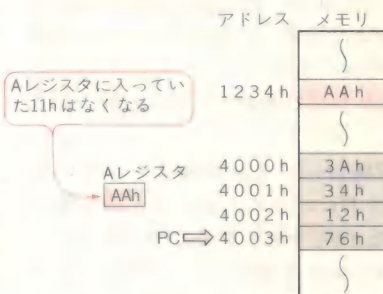
〈図6〉 I/O リード/ライト・サイクル



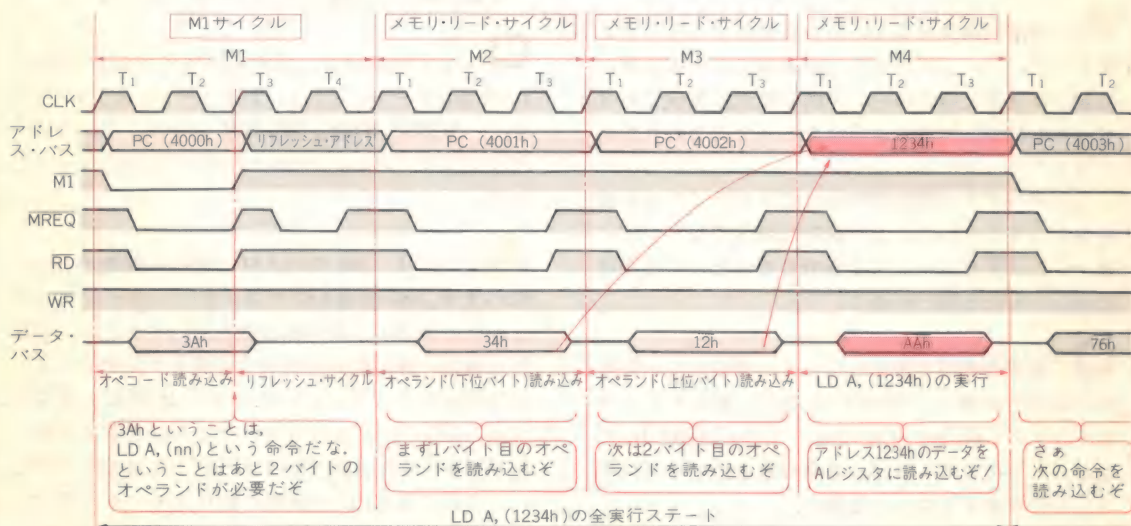
〈図7〉 LD A, (nn) 命令の実行のようす



(a) 実行前のメモリとレジスタ



(c) 実行後のメモリとレジスタ



(b) バスの動作のようす



に行われます。図6にI/Oリード/ライト・サイクルのタイミングを示します。

I/Oリード・サイクルでは、 $T_1$ のときにI/Oアドレスが $A_0 \sim A_7$ (正確には $A_0 \sim A_{15}$ )にのせられ、 $T_2$ の立ち下がりでは $\overline{IORQ}$ と $\overline{RD}$ がLレベルになります。

I/Oから読み取られるデータは $T_3$ の立ち下がりを取り込まれます。また、I/Oライト・サイクルでも $T_1$ のときにI/Oアドレスが $A_0 \sim A_7$ (正確には $A_0 \sim A_{15}$ )にのせられ、I/Oに書き込むデータが $D_0 \sim D_7$ 上にのせられます。そして、 $T_2$ の立ち下がりでは $\overline{IORQ}$ と $\overline{WR}$ がLレベルになります。

両マシン・サイクルとも $T_2$ と $T_3$ の間に $T_w$ というステートがありますが、これはCPUが自動的に挿入するウェイト・ステートで、あまりアクセス・タイムの速くないMOSのI/Oなどに対処するためです。

### ● マシン・サイクルの実行例

ここでは以上で説明した事がらをさらにわかりやすくするため、実際の命令を実行する場合のマシン・サイクルの動きについて見てみます。とりあげる命令は、

- (1) LD A, (nn)
  - (2) LD (nn), A
  - (3) IN A, (n)
  - (4) OUT (n), A
- です。

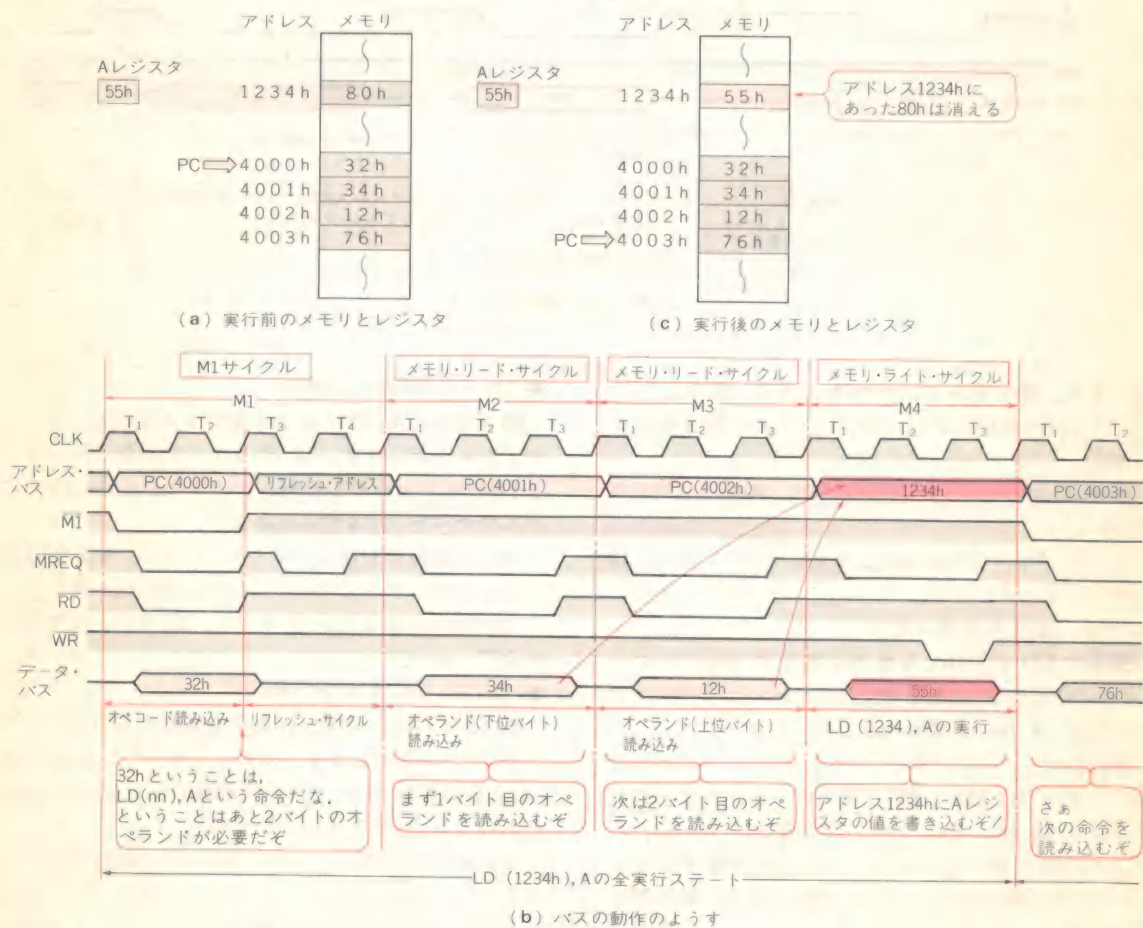
ここでは、説明をより現実的なものにするため、nn, nならびにアキュムレータ(Aレジスタ)の値をそれぞれ1234h, 80h, 55hとします。また、メモリおよびI/Oからデータを読み出す場合、読み出されるデータはAAhとします。さらに、これらの命令はすべて4000h番地から始まるものとします。

#### ▶ LD A, (nn)命令の実行

図7にLD A, (1234h)を実行する場合のマシン・サイクルを示します。M1サイクルでオペコード、3Ahをフェッチします。この命令は二つのオペランドをもつので、次にメモリ・リード・サイクルが2回続きます。これで読み出すアドレスが確定します。

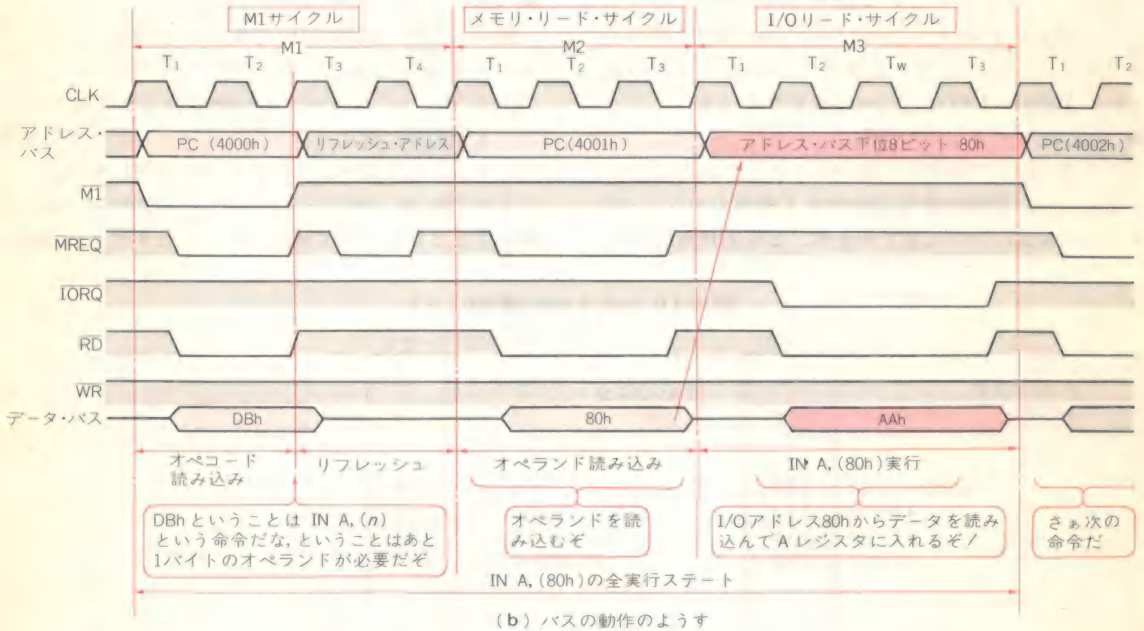
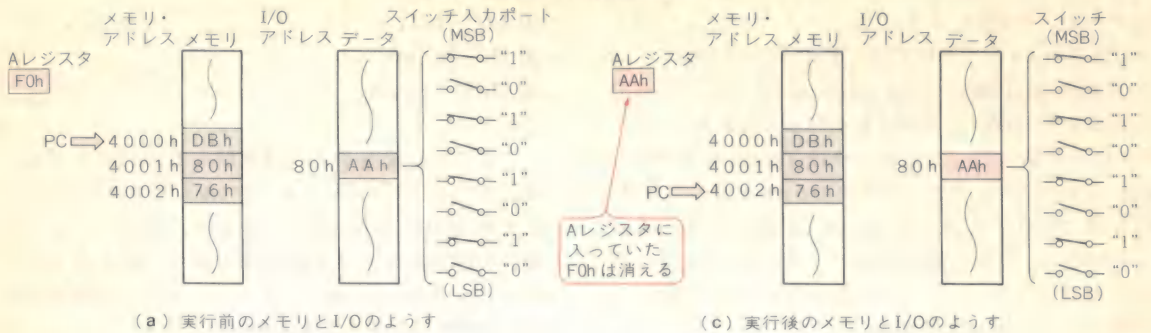
またZ80は、16ビットのアドレスやデータを読み書きするときは、下位8ビット→上位8ビットの順で

〈図8〉LD (nn), A命令の実行のようす





〈図9〉 IN A, (n)の実行のようす



行います。図7をみるとわかるように、読み出しアドレス 1234h を得るのに、下位バイトから読み込んでいます。

最後のメモリ・リード・サイクルが、実際にアドレス 1234h からメモリ・データをアキュムレータに読み込むところです。この命令は、4 マシン・サイクル、13 ステートで完了します。

#### ▶ LD (nn), A 命令の実行

図8に LD (1234h), A 命令を実行する場合のマシン・サイクルを示します。初めの三つのマシン・サイクルは LD A, (nn) 命令の場合と同じです。

最後のサイクルはメモリ・ライト・サイクルです。このサイクルでは M2 と M3 で読み取ったオペランドをアドレスとして、そのメモリ・アドレスにアキュムレータの内容を書き込みます。この命令が要するマシン・サイクル数は4で、ステート数は13と、前の例と同じです。

#### ▶ IN A, (n) 命令の実行

図9に IN A, (80h) 命令を実行する場合のマシン・サイクルを示します。M1 サイクルで命令のオペコード DBh をフェッチした後、次のメモリ・リード・サイクルでオペランドを読み込みます。

そして最後の I/O リード・サイクルで、その I/O ポートからデータをアキュムレータに読み込みます。

この命令は3マシン・サイクル、11ステートで実行が完了します。

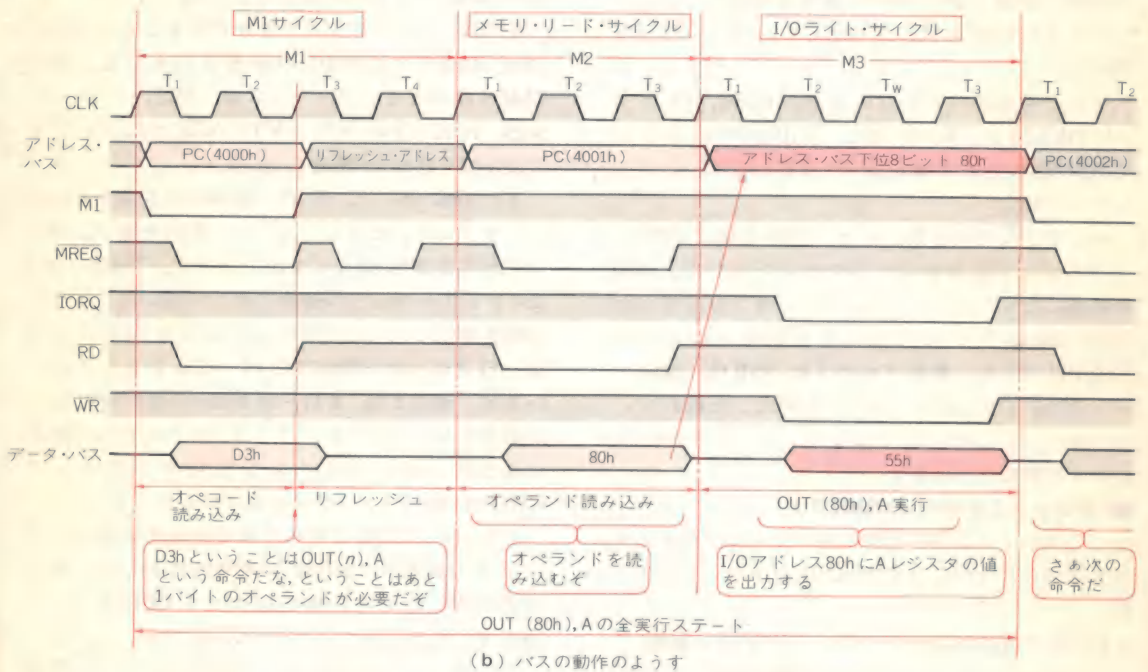
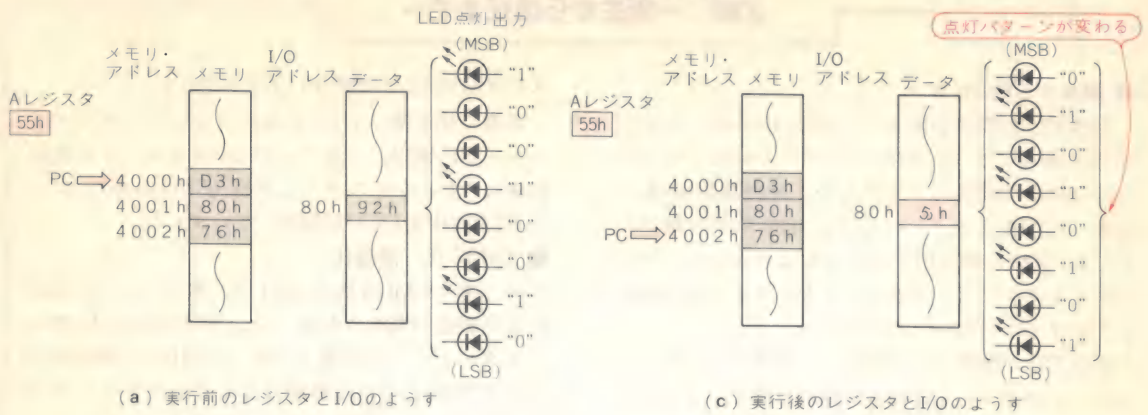
#### ▶ OUT (n), A 命令の実行

図10に OUT (80h), A 命令を実行する場合のマシン・サイクルを示します。初めの2サイクルは IN A, (n) 命令の場合と同じですが、最後のマシン・サイクルは I/O ライト・サイクルになります。

このサイクルでは、アキュムレータの内容を命令のオペランドで示されるアドレスの I/O ポートに書き込みます。命令の実行に必要なマシン・サイクルとス



〈図 10〉 OUT (n), A の実行のようす



テート数は IN 命令の場合と同じです。

### ● Z80 の割り込みシステム

マイコン・システムにおいても一つ重要な概念があります。第 6 章で詳しく解説しますが、割り込みという概念です。

Z80 ではこの割り込みについても、Z80 ファミリーと呼ばれる Z80 用に作られた周辺 LSI と組み合わせることで、非常に強力な割り込みシステムを構築できます。

\* \*

### ● おわりに

筆者は 10 年ほど前に著した書籍の冒頭に、「Z80 は

永遠である」と書きましたが、永遠とはいかなくても、現在まで Z80 が建在なのを知り、本当に喜んでいま

す。  
4040 に始まり、8080A、6800、8085、Z80、8086、80x86 と種々の CPU と付き合いってきましたが、Z80 は今でも心に残る CPU です。

### ●参考文献●

- (1) VOLUME 1 DATEBOOK, MICROPROCESSORS AND PERIPHERALS, ZILOG.
- (2) Z80 Family User's Manual, ZILOG.

(トランジスタ技術 1994 年 6 月号に加筆、修正)



### ● 誕生から約 20 年

Z80CPU の開発が始まったのは 1974 年、今から 20 年も前のことで、初めてデバイスが世の中に出たのはその 2 年後のことでした。発売開始当時、1 個数十万円もしたデバイスで、1980 年頃の雑誌などでも「Z80 が数万円で手に入るようになり、やっと使えるようになって来た…」というくだりがあるようなマイコンだったのです。

Z80CPU が開発された頃は、マイクロコンピュータという物がやっと認知され始めた頃で、インテルが 4004、8008 に続き 8080 の量産を開始し、またモトローラが 6800 を世に送り出してしばらくたった頃です。

メモリも 512 バイトのヒューズ ROM、1K ビットの DRAM が出始めた頃で、入出力デバイスとして ASR33 などの正真正銘の「TTY 端末」が存在していた頃です。

このような時代背景のもと、8080 を超え 8080 とソフトウェア互換性をもつマイコンとして、8080 を開発したエンジニア 4 人がスピンオフして作った会社、Zilog(ザイログ)社により開発されたのが Z80CPU でした。製造プロセスは NMOS、2933 ゲート(8933 トランジスタ相当)を使用した設計で、命令はマイクロ・コードではなく、ハードワイヤ・ロジックで組まれています。

### ● クロック 2.5 MHz の Z80CPU

このデバイスの発売当初の謳い文句は、今からすると信じられないことですが、次のようなものでした。

- ・5 V 単一電源(8080 は±5 V、+12 V の 3 電源)
- ・シングルフェーズ・クロック(8080 は 2 相クロック)
- ・強化された割り込み(モード 1、2 割り込み)
- ・増強されたレジスタ・セット(裏レジスタおよびインデックス・レジスタ)
- ・強化された 158 種の命令(8080 は公称 78 種類)

発売当初、プロセスは NMOS、動作クロックは 2.5 MHz でした。その後、クロックも 4 MHz(その当時は A バージョンと呼ばれていた)、6 MHz(同 B バージョン)とクロックも早くなっていき、1984 年には H バージョンと呼ばれるクロック 8 MHz の製品も発表されました。

いっぽうの周辺デバイスは、1970 年代後半にはすべてそろい、CPU クロックの高速化に歩調をあわせ、4 MHz、6 MHz のバージョンが世に送り出されていきました。残念ながら NMOS 周辺デバイ

スの 8 MHz 品は開発されませんでした。

筆者の知る限り NMOS 品のセカンド・ソースは、シャープ、SGS、モステック(SGS トムソンに吸収合併)、ゴールドスター、NEC(CPU のみ)、ローム(現在は中止)などが製造していました。

### ● CMOS 化、高速化

いっぽう CMOS 版は 1984 年、東芝との技術提携により製造が開始されました。当初は 6 MHz 版からスタートし、その後 8、10、20 MHz と高速化し、また VTI 社との技術提携により Z80 ファミリすべてが ASIC ライブラリ化されました。

その後、このライブラリを元にさまざまな ASSP(特定用途向け汎用 IC)が発表されました。現在 CMOS 版のセカンド・ソースは、東芝、シャープ、SGS、NEC(CPU のみ)、VTI(ASIC のコアとしての使用権を持つ)が製造しています。

また 1987 年には、日立が HD64180 を開発・発表し、ザイログとセカンド・ソース契約を結び、ザイログからは Z64180 として出ています。Z64180 では Z80 ファミリ周辺との接続に若干問題があり、その問題を解決したものをザイログからは Z180 として、また日立からは HD64180Z バージョンとして出ています。現在 6 M、8 M、10 MHz の各バージョンが用意されていますが、ザイログからはさらに高速化を図り、またデザインを完全スタティック CMOS に直したバージョン 8S180 が出ています。

そのほか、Z180 を中心に周辺機能を集積したデバイス Z80181 や Z80182 も開発されました。そして Z80CPU の上位互換の 16 ビット CPU として、1987 年に販売開始されたのが Z280 です。

このデバイスは Z800 として開発されていた製品で、製品概要が発表されてからなかなか現物が出てこなかったため、「嘘八百」と陰口を叩かれていた製品でもありました。

そのアーキテクチャから、Z80CPU 用に書かれたアプリケーションをそのまま走らせた場合、それほどパフォーマンスが上がらなかったことから、幅広く採用されるまでには至りませんでした。

そして現在、その欠点を克服し 16/32 ビット・アーキテクチャで新たに設計されたデバイスが 1993 年に発表された Z380 です。

また R800(アスキー三井セミコンダクター)や KL5C8012(川崎製鉄)などのような Z80 ソフトウェア互換の高速 CPU も登場しています。

〈信垣育司〉



## Z80を自由自在に動かすための

# Z80のレジスタと命令

野口智樹

### Z80 のレジスタ

#### ● レジスタとは

レジスタとはお店にあるレジと同じ意味で、値を格納するためのものです。値を格納するだけならメモリ(RAM)と同じですが、メモリに対しての値の格納よりも高速かつ簡単に操作が行えます。

Z80 にはいろいろな種類のレジスタが用意されています。第1章でも少し触れましたが、ここでは特にソフトウェア的なところから解説します。図1にZ80のレジスタ・セットを示します。

#### ▶ フラグ・レジスタ (F)

フラグ・レジスタそのものは8ビットのレジスタですが、その内容の1ビットごとがそれぞれのフラグとしての意味をもっています。

フラグとは演算などの実行結果の状態を示すもので

す。フラグ・レジスタに関する詳細は後述します。

#### ▶ 汎用レジスタ (A, B, C, D, E, H, L)

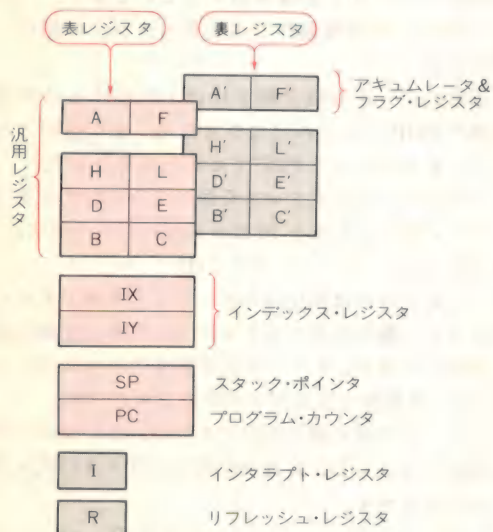
汎用レジスタとは特に専用の目的がなく、プログラムで自由に使用することのできるレジスタのことです。

それぞれAレジスタ、Bレジスタ、Cレジスタ...と呼びますが、すべて8ビット(1バイト)のレジスタです。特にAレジスタ以外のB、C、D、E、H、Lは、BC、DE、HLといった具合に二つを組み合わせで16ビット(2バイト)の値を扱えます。これらを**ペア・レジスタ**といいます。

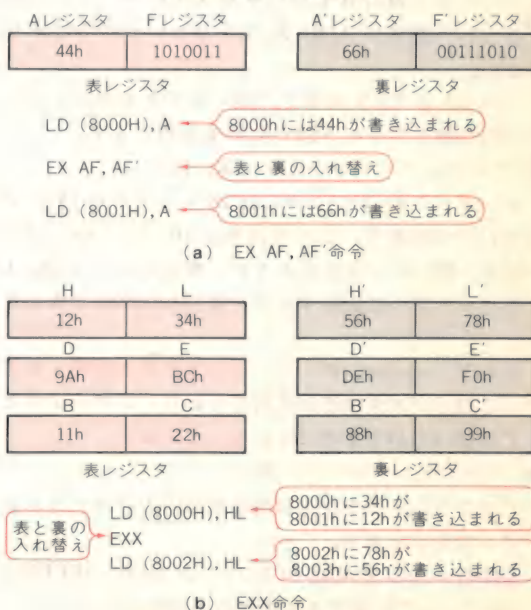
Aレジスタは前述のフラグ・レジスタ(F)とペアになりますが、AFの形では16ビットの値は扱えません(PUSH/POP/EXなどの命令でペアになる)。

また、特に専用の目的がないと説明しましたが、Bレジスタが8ビットのカウンタに、CレジスタがI/Oポートのアドレス指定に、それぞれ使われる専用の命令も一部にはあります。

〈図1〉 Z80のレジスタ・セット

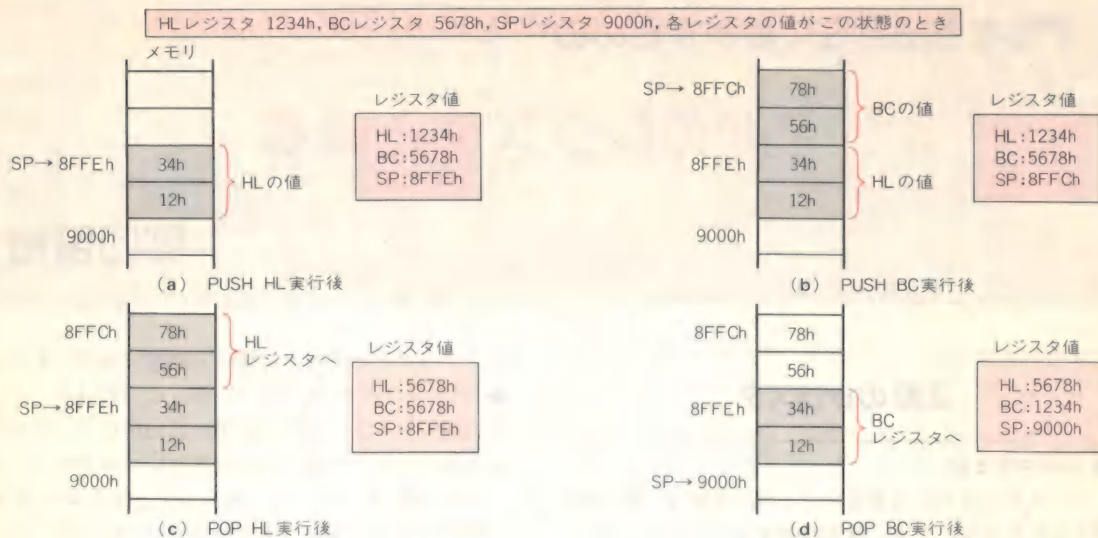


〈図2〉 EX と EXX 命令の動作





〈図3〉 PUSH, POP の動作



さらに汎用レジスタのうち、特にAレジスタは**アキュムレータ**と呼ばれ、ADD/SUB/AND/OR/XORなどの演算結果を直接格納できる唯一のレジスタです。

#### ▶ 裏レジスタ (AF', BC', DE', HL')

Z80では前述したフラグ・レジスタと汎用レジスタと同様のものをもう1組もっており、これを裏レジスタといいます。裏レジスタだからといって使い方に特に違いはありません。

EX AF, AF'命令でA, Fレジスタを、EXX命令でB, C, D, E, H, Lレジスタの裏と表を切り替えることができます(図2)。

このように切り替えて使用するので、裏レジスタのBレジスタと表のAレジスタをダイレクトに足し算するという意味のADD A, B'などという命令はありません。

このときはEXX命令で、裏レジスタを表にレジスタに切り替えてからADD A, Bを実行します。

#### ▶ インデックス・レジスタ (IX, IY)

インデックス・レジスタにはIXレジスタとIYレジスタの二つがあり、これらは共に16ビットのレジスタです。HLレジスタのように、場合によってH, Lと8ビットずつに使用することは(正式には)できません。

またアドレッシング・モードのところで説明しますが、このレジスタはその名前のとおりインデックスとしての使い方ができるようになっています。

#### ▶ プログラム・カウンタ(PC)

プログラム・カウンタは現在実行中のプログラムのアドレスを示します。

Z80のメモリ空間は64Kバイト(0000~0FFFFh)と16ビットで表現できますから、当然PCは16ビット

レジスタです。

#### ▶ スタック・ポインタ(SP)

スタックとはひとことではいうと先入れ後出しの一時格納庫のことです。

スタック操作命令の代表としてPUSH/POP命令があります。例えばPUSH BCと実行すればBCの内容が一時的にスタックというデータ領域に格納されます。これを再度取り出すにはPOP BCとします。

ここでPUSH HL, PUSH BCを実行したあと、POP HL, POP BCを実行すると、BCとHLの内容が入れ替わってしまいます。図3をみてください。

このように、先にPUSHしたデータの上に次にPUSHするデータが積み重なっていきます。このイメージからPUSH動作のことをスタックに積むとも呼びます。

正常にデータを取り戻すには、PUSHとは反対の順番でPOPしなければなりません。また値が入れ替わることをねらってPOPすることもあります。

このスタックのアドレスを保持するのがスタック・ポインタです。これも16ビットのレジスタです。

#### ▶ 割り込みベクタ・レジスタ(I)

このレジスタは割り込みモード2で意味のあるレジスタです。割り込みシステムについては第6章で詳しく説明しますが、Iレジスタは割り込みベクタの上位8ビットを保持する8ビットのレジスタです。

このレジスタに値を設定するには、Aレジスタに値を設定してからLD I, A命令でIレジスタに代入するしかありません。

#### ▶ メモリ・リフレッシュ・レジスタ(R)

本来はDRAMのリフレッシュ制御のためのレジスタですが、現在では実際にそのような用途で使用する



〈表 1〉 キャリ・フラグとゼロ・フラグの動作

実行前の A レジスタ	命令	実行後の A レジスタ	実行後のフラグ	
			キャリ・フラグ	ゼロ・フラグ
FFh	SUB 1	FEh	0	0
00h	SUB 1	FFh	1	0
01h	SUB 1	00h	0	1
02h	SUB 1	01h	0	0
FEh	ADD A.1	FFh	0	0
FFh	ADD A.1	00h	1	1
00h	ADD A.1	01h	0	0
01h	ADD A.1	02h	0	0

ことはまずありません。

このレジスタは8ビットですが、このうち最上位ビット(ビット7)の状態は変化しません。下位7ビットが規則的に刻々と変化するレジスタであり、この性質を利用して乱数の生成などで使用することもできます。

## Z80のフラグの動作

### ● フラグの役割

CPUは演算を行った結果、判定を行うために各種のフラグが用意されています。フラグとは“1”か“0”の二つの状態のいずれかを示すものです。

Z80には8ビットのフラグ・レジスタのうち、6ビットが意味のあるフラグとして割り当てられています。

6ビットのフラグのうち、特に重要なのは**ゼロ・フラグ(Z)**、**キャリ・フラグ(C)**、**サイン・フラグ(S)**、**パリティ/オーバ・フロー・フラグ(P/V)**の四つです。

これらのフラグがどのビットであるかということは重要ではありませんが、各フラグの示す意味や演算結果によるフラグの変化については、しっかりと理解しておく必要があります。

### ● キャリ・フラグ(C)

これは**演算の結果、桁あふれが生じたことを示すフラグ**です。キャリ・フラグは符号なし演算の場合に意味があります。

例えば、8ビットの符号なし数値の最大値は0FFh(255)ですが、加算命令などでこれを越えるような演算の結果が生じた場合にキャリ・フラグが1になります。また、それとは逆に減算命令などで0より小さい結果が生じた場合にも同様にキャリ・フラグが1になります。

キャリ・フラグは、ローテイト・シフト命令などでも使われます。

なお、キャリ・フラグは、SCF命令でセット(1)、CCF命令でリセット(0)と直接コントロールすることもできます。

### ● ゼロ・フラグ(Z)

これは**演算結果がゼロであることを示すフラグ**です。

〈表 2〉 サイン・フラグの動作

符号なしの値	符号付きの値	サイン・フラグ	2進数表現
0	0	0	00000000
1	1	0	00000001
2	2	0	00000010
⋮	⋮	⋮	⋮
127	127	0	01111111
128	-128	1	10000000
129	-127	1	10000001
⋮	⋮	⋮	⋮
254	-2	1	11111110
255	-1	1	11111111

例えば、いまAレジスタに1が入っているとしてDEC A(Aを-1する)命令を実行するとAレジスタは0になり、ゼロ・フラグは1になります。

逆に現在Aレジスタに0が入っているとしてINC A(Aを+1する)命令を実行するとAレジスタは1になりますが、ゼロ・フラグは0になります。

なお、キャリ・フラグは、SCF/CCF命令で直接セットしたり反転したりすることができ、ゼロ・フラグに対してはこのような特別な命令は用意されていません。

表1にキャリ・フラグとゼロ・フラグの動作について示します。

### ● サイン・フラグ(S)

これは**符号(+/-)の状態を示すフラグ**です。

当然、このフラグは符号付きの演算をしている場合に使われます。符号付きとは2の補数の表現をした値のことです。8ビットのある演算結果を例として示すと、表2のようになります。

このように符号付き表現では最上位ビットが符号を示します。サイン・フラグは、0で正を、1で負を示します。つまりサイン・フラグとは、演算結果の最上位ビットをそのままフラグにコピーしたものとも考えることもできます。

### ● パリティ/オーバ・フロー・フラグ(P/V)

これは**1になっているビット数が偶数個あるか奇数個あるかのパリティを示したり、オーバ・フローを示すフラグ**です。命令によってどちらの意味になるかが決められています。

論理演算命令(AND/OR/XOR/ローテイト・シフト/IN r, (C)/DAA など)の場合は、パリティ・フラグとしての意味をもちます。

算術演算命令(ADD/ADC/SUB/SBC/INC/DEC など)の場合は、オーバ・フロー・フラグとしての意味をもちます。

このフラグの示すオーバ・フローとは、符号付きの演算をしている場合のことです。符号なしの演算のオーバ・フローはキャリ・フラグを用います。



## Z80のアドレッシング・モード

### ● アドレッシング・モードとは

アドレッシング・モードとは、CPU がある命令を実行する場合のパラメータ構造について規定したものです。ここではそのアドレッシング・モードについて、一つ一つ簡潔に説明していきます。表3に各種アドレッシング・モードをまとめます。

### ● アドレッシング・モードのいろいろ

#### ▶ イミディエイト・モード (Immediate)

このモードは、その命令が対象とする1バイトのオペランドをもつ場合のアドレッシング・モードです。

イミディエイト・モードなどと難しそうな名前がついていますが、はやい話が指定した値をそのままレジスタやメモリに代入することです。

#### ▶ 拡張イミディエイト・モード (Extended Immediate)

このモードはその名のとおりイミディエイト・モードの拡張で、1バイトではなく2バイトのオペランドをもつ場合のアドレッシング・モードです。

イミディエイト・モードとは8ビットで、拡張イミディエイト・モードが16ビットと覚えましょう。

#### ▶ 拡張アドレス・モード (Extended)

これは2バイトからなるオペランド・アドレスをもつアドレッシング・モードです。指定したアドレスの1バイトあるいは2バイトのデータに対して操作をするモードです。

#### ▶ インデックス・モード (Index)

これはインデックス・レジスタ (IX, IY) を用いる場合に使われます。

インデックス・モードでは、インデックス・レジスタ (IX, IY) の値とディスプレースメント (+d/-d) の和が使用されます。

ディスプレースメントとは LD A, (IX+d) の d のことです。この d は -128(80h) から +127(7Fh) の範囲で指定可能です。

#### ▶ 間接アドレッシング・モード (Indirect)

Z80 の場合には HL, DE, BC などベア・レジスタによりメモリ・アドレスを指定することをいいます。

例えば LD A, (HL) などがその代表です。これは HL レジスタで示されるアドレスの内容を A レジスタに代入するという命令です。

#### ▶ レジスタ・モード (Register)

レジスタからレジスタへのデータ転送や演算など、レジスタどうしの操作の場合のモードです。これはオペコードの中に対象となるレジスタを指定するビットをもつ命令に適用されます。

#### ▶ インプライド・モード (Implied)

〈表3〉アドレッシング・モード一覧

アドレッシング・モード	機械語コード (16進数)	アセンブリ言語	
イミディエイト・モード	3E 01 06 CB	LD	A, 1 B, 0CBH
拡張イミディエイト・モード	01 34 12 DD 21 AB 89	LD	BC, 1234H IX, 89ABH
拡張アドレス・モード	3A 00 80 2A 34 12	LD	A, (8000H) HL, (1234H)
インデックス・モード	DD 7E 60 FD 36 FF 99	LD	A, (IX+56H) LD (IX-1), 99H
間接アドレッシング・モード	77 ED 40	LD	A, (HL) IN B, (C)
レジスタ・モード	41 C5	LD	B, C PUSH BC
インプライド・モード	C6 50 90	ADD	A, 50H SUB B
ビット・モード	CB D7 CB 81 CB 7B	SET	2, A RES 0, C BIT 7, E
相対アドレス・モード	18 FD	JR	LOOP1
ページ・ゼロ・モード	C7 FF	RST	00H RST 38H

Z80 の8ビット演算命令 (ADD/SUB/AND/OR/XOR/CP) では、A レジスタの値と何らかのデータという形で必ず A レジスタが使われ、演算結果も常に A レジスタに格納されます。よってこれらの命令には、A レジスタのオペランド指定がなくても、A レジスタを使うことがわかりきっています。

この A レジスタのオペランド指定が省略されたモードが、インプライド・モードです。

しかし、ADD, ADC, SBC 命令だけは16ビット演算との区別のために、ADD A, B などのように A レジスタの記述が必要です。

#### ▶ ビット・モード (Bit)

これは BIT/SET/RES 命令がもつアドレッシング・モードです。これらの命令は、何レジスタの何ビット目をセット (1) するか、リセット (0) するか、または 0 か 1 かをチェックする命令です。

#### ▶ 相対アドレス・モード (Relative)

これは JR/DJNZ 命令だけがもつアドレッシング・モードです。JR/DJNZ (相対ジャンプ) 命令は、JP (絶対ジャンプ) 命令と異なり、ジャンプ先を現在位置からいくつ先というように相対的に表現します。

1バイトの相対アドレスの範囲は -128(80h) から +127(7Fh) で、当然この範囲を越えるような相対ジャンプはできません。

ただしアセンブリ言語で記述するときには、ジャンプ先が現在のロケーション・カウンタからどれだけ足した、または引いたアドレスかなどと意識する必要はありません。通常の JP 命令と同様にジャンプ先のラ



ベルを指定するだけです。この相対アドレスの計算はアセンブラがやってくれます。

もしそのジャンプ先が-128～+127の範囲に入らない所にあった場合はアセンブラがエラーを出してくれるので、そのときにJP命令に直せばよいと気楽に考えて大丈夫です。

#### ▶ ページ・ゼロ・モード (Page Zero)

これは **RST 命令のみ**がもつアドレッシング・モードです。

ページ・ゼロとは 0000h～00FFh の 100h バイトの領域のことで、RST 命令はオペランドに 00H, 08H, 10H, 18H, 20H, 28H, 30H, 38H の 8 通りを受け入れます。

RST 命令は 1 バイト命令のため機械語では、これらのアドレスを 1 バイト中の 3 ビット (ビット 3～5) で表現しています。

#### ● アドレッシング・モードのまとめ

以上、アドレッシング・モードについて説明しましたが、細かいことにこだわらずに別の表現で要約してみると以下ようになります。

▶ 直接値レジスタに代入するときはそのまま値を書く

例: LD HL, 1234H HL レジスタに 1234h を代入

▶ ( ) で囲まれたものは、そのアドレスのデータの意味

例: LD HL, (1234H) L レジスタに 1234h のアドレスのデータを、H レジスタに 1235h のアドレスのデータを代入

例: LD A, (DE) A レジスタに DE レジスタの値で示されるアドレスのデータを代入

▶ インデックス・レジスタ (IX/IY) は、(IX+nnn) もしくは (IX-nnn) の形で -128～+127 の範囲をアクセスできる。

例: LD A, (IX+5) IX レジスタの値に 5 を足したアドレスのデータを A レジスタに代入

## よく使う Z80 の命令

Z80 には 158 種類もの命令がありますが、これをすべて覚えておく必要はありません。その中でもよく使う命令は限られています。

ここでは紙面の都合もあり、特によく使う命令について解説します。これだけ覚えておけば、ほとんどの Z80 のプログラムは大丈夫でしょう。

#### ● データの転送 (リスト 1)

Z80 ではデータの転送は、レジスタ-レジスタでも、レジスタ-メモリでも、すべて LD 命令を使います。また代入方向は、LD A, B なら、B レジスタの内容を A レジスタに代入するというように、右から左です。

#### 〈リスト 1〉 データ転送命令の例

LD	E, B	:B レジスタから E レジスタへ
LD	A, (HL)	:HL のアドレスのメモリを A レジスタへ
LD	(BC), A	:A レジスタを BC のアドレスへ
LD	A, (IX+100)	:IX+100 のアドレスのメモリを A レジスタへ
PUSH	HL	:HL レジスタのデータをスタックに積む
POP	DE	:スタックから DE レジスタにデータを取り出す
LD	HL, 8000H	:HL レジスタに 8000H を代入
LD	DE, 9000H	:DE レジスタに 9000H を代入
LD	BC, 1000H	:BC レジスタに 1000H を代入
LDIR		:ブロック転送命令
EXX		:BC, DE, HL レジスタと BC', DE', HL' レジスタ交換
EX	AF, AF'	:A, F レジスタと A', F' レジスタを交換
EX	DE, HL	:DE レジスタと HL レジスタを交換

#### ▶ レジスタ間転送

汎用 8 ビット・レジスタ間の転送はどのレジスタとの組み合わせでも可能です。

16 ビット・レジスタ間の転送命令は、LD SP, HL/ LD SP, IX/ LD SP, IY の三つしかありません。汎用 16 ビット・レジスタの転送は、ペア・レジスタを 8 ビットずつに分けて 2 命令で実行します。

またインデックス・レジスタのように 8 ビットに分割できないレジスタは、PUSH 命令でスタックに積んでから POP 命令で読み出します。

#### ▶ レジスタ-メモリ間

8 ビットなら A レジスタで、16 ビットなら HL, IX, IY レジスタで使えます。また間接アドレッシング・モードを使えば、BC レジスタや DE レジスタで示されるアドレスのメモリ内容を A レジスタに転送することもできます。

#### ▶ PUSH/POP 命令

すでに説明したように、スタック領域に値を一時的に待避するときに使用する 16 ビット専用命令です。A レジスタの待避はフラグ・レジスタと一緒に AF という形で行います。

また PUSH HL を実行したあとに POP DE を実行すると、HL レジスタの内容を DE レジスタに転送することにもなります。

#### ▶ ブロック転送命令

HL レジスタを転送元の先頭アドレスに、DE レジスタを転送先の先頭アドレスにして、BC レジスタで示されるバイト数だけ転送する命令が、**LDIR 命令**です。1 バイト転送するごとに、HL と DE レジスタは +1 し、BC レジスタは -1 されます。

これとは逆に、転送元の終了アドレスを HL レジスタに、転送先の終了アドレスを DE レジスタにして、BC レジスタのバイト数だけ転送する命令が **LDDR 命令**です。こちらは 1 バイト転送するごとに、HL, DE, BC レジスタを -1 していきます。

#### ▶ 交換命令

Z80 には裏レジスタがあることは説明しました。こ



## 〈リスト2〉 演算命令の例

```

INC  A      ;A = A +1
INC  BC     ;BC=BC+1
DEC  E      ;E = E -1
DEC  IY     ;IY=IY-1
ADD  A,B    ;A = A +B
ADD  HL,DE  ;HL=HL, DE
ADC  A,B    ;A = A +B +Cy(キャリ・フラグ)
ADC  HL,BC  ;HL=HL+BC+Cy(キャリ・フラグ)
SUB  D      ;A = A -D
SBC  A,E    ;A = A -E -Cy(キャリ・フラグ)
SBC  HL,DE  ;HL=HL-DE-Cy(キャリ・フラグ)
CP   B      ;AレジスタとBレジスタを比較
CP   55H    ;Aレジスタと55hを比較
AND  C      ;AレジスタとCレジスタをANDを取る
XOR  0AAH   ;Aレジスタと0AAhの排他的論理和を取る

```

の切り替えのために EXX 命令が用意されています。

**EXX 命令**は HL, DE, BCレジスタの切り替え用、**EX AF, AF'命令**が A レジスタとフラグ・レジスタの交換命令です。

さらに DE レジスタと HL レジスタの内容を切り替える **EX DE, HL** という命令もあります。

また、交換命令そのものによるフラグの変化はありませんが、EX AF, AF'命令については、裏レジスタだったフラグ・レジスタに切り替わるわけですから注意が必要です。

## ● 演算命令(リスト2)

### ▶ INC/DEC 命令

レジスタの内容を+1する命令が INC、-1する命令が DEC です。汎用レジスタやインデックス・レジスタで使用できます。またフラグの変化も注意が必要で、8ビットでは INC 命令で 0 になっても、また DEC 命令で 255(0FFh)になってもキャリ・フラグは変化しません。また 16ビット・レジスタの INC/DEC 命令では、ゼロ・フラグも変化しません。

### ▶ ADD 命令

Z80 では INC/DEC 命令以外の演算命令は、8ビットなら A レジスタに、16ビットなら HL レジスタに集中しています。つまり、一度 A レジスタに数値を入れてから計算する必要があります。

まず加算命令としては ADD 命令があります。8ビットなら A レジスタと、16ビットなら HL レジスタ、もしくはインデックス・レジスタとの足し算になります。桁あふれがあるとキャリ・フラグが 1 になります。

### ▶ ADC 命令

これも加算命令なのですが、キャリ・フラグの内容も加算する命令です。命令実行前のキャリ・フラグが 1 のとき、足し算結果にさらに 1 が加算されます。実行前のキャリ・フラグがゼロなら、ADD 命令と同じ演算結果になります。フラグの変化も ADD 命令と同じです。

### ▶ SUB 命令

## 〈リスト3〉 分岐命令の例

```

JP   8000H   ;8000h番地にジャンプ
JR   9800H   ;8000h番地にジャンプ(相対ジャンプ)
JP   C, 0200H ;キャリ・フラグが1なら200hにジャンプ
JR   NC, 0100H ;キャリ・フラグが0なら100hにジャンプ
JP   NZ, 4000H ;ゼロ・フラグが0なら4000hにジャンプ
JR   Z, 4100H ;ゼロ・フラグが1なら4100hにジャンプ
CALL 0800H   ;サブルーチン 800hをコール
...
RET         ;リターン

```

8ビット専用命令で、A レジスタからオペランドの内容を引き算します。結果が 0 ならゼロ・フラグが 1 に、オペランドの値より A レジスタのほうが小さければキャリ・フラグが 1 になります。

### ▶ SBC 命令

これは ADD 命令に対する ADC 命令と同じで、この命令の実行前のキャリ・フラグが 1 であれば、引き算した結果からさらに 1 を引きます。フラグの変化は SUB 命令と同じです。実行前のキャリ・フラグがゼロなら SUB 命令と同じ演算結果になります。

16ビットの引き算はこの命令しかないので、実行前にキャリ・フラグをリセットしてからこの命令を実行します。

### ▶ CP 命令

8ビット専用の引き算命令ですが、実際に引き算した値は A レジスタには格納されません。フラグだけが変化します。フラグの変化は SUB 命令と同じです。これは値を比較したいときに使用します。

16ビット・レジスタの比較をしたいときは、いったんレジスタを保存してから、キャリ・フラグをリセットして SBC 命令を実行するしかありません。

### ▶ AND, OR, XOR 命令

論理演算命令で、8ビット専用です。論理演算命令では、キャリ・フラグはリセットされます。

## ● 分岐命令(リスト3)

### ▶ JP/JR 命令

プログラムの実行の流れを変える無条件分岐命令です。フラグは変化しません。JP は絶対アドレス、JR は相対アドレス・ジャンプ命令です。

### ▶ JP C, ???/JR C, ???/JP NC, ??/JR NC, ??命令

条件分岐命令です。キャリ・フラグが 1 ならオペランドのアドレスにジャンプします。0 ならジャンプせず次の命令を実行します。NC がつくとフラグが 0 ならジャンプです。

### ▶ JP Z, ???/JR Z, ???/JP NZ, ??/JR NZ, ??命令

条件分岐命令で、ゼロ・フラグが 1 ならオペランドのアドレスにジャンプします。0 ならジャンプせず次の命令を実行します。NZ がつくとフラグが 0 ならジャンプします。



## ▶ CALL/RET 命令

サブルーチン・コール命令です。サブルーチンから戻るには RET 命令を使います。

## ▶ DJNZ 命令

プログラムにおいてループ処理はよく出てくる処理です。この命令は B レジスタをループのカウンタとして使う命令で、B レジスタを-1 して、0 以外なら指定アドレスにジャンプし、0 なら次の命令を実行します。フラグは変化しません。

この命令は相対アドレス・ジャンプになるので、ジャンプ先が-128~-+127 バイトの範囲外のときは使えません。

## ● 入出力命令(リスト 4)

Z80 にはメモリ空間と I/O 空間があることは説明しました。LD 命令はメモリ空間に対するアクセスでした。I/O 空間に対するアクセスには次の命令が用意さ

## 〈リスト 4〉 入出力命令の例

IN	A, (80H)	: I/O アドレス 80h のデータを A レジスタに入力
IN	B, (C)	: C レジスタの I/O アドレスから B レジスタに入力
OUT	(90H), A	: I/O アドレス 90h に A レジスタを出力
OUT	(C), E	: C レジスタの I/O アドレスに E レジスタを出力
LD	HL, 9000H	: HL レジスタに 9000h を代入
LD	B, 16	: B レジスタに 16 を代入
LD	C, 1CH	: C レジスタに 1Ch を代入
INIR		: ブロック入力(16 バイト I/O アドレス 1Ch)
LD	HL, 8000H	: HL レジスタに 8000h を代入
LD	B, 32	: B レジスタに 32 を代入
OTIR		: ブロック出力(32 バイト I/O アドレス 1Ch)

れています。

## ▶ IN 命令

オペランドで指定された I/O アドレスからデータを読み込み、A レジスタに転送します。また入力に 8 ビット汎用レジスタを使いたいときは、C レジスタを I/O アドレスとして、IN D, (C) のようにも使えます。

## Z80 七不思議 レジスタ・命令編

**Q** : EX DE, HL はあるのに EX BC, HL がないのはなぜ?

**A** : これは単に 8080A で EX DE, HL に相当する命令はある(XCHG)が、EX BC, HL に相当する命令がない、ということではないでしょうか。

**Q** : なぜ R レジスタが 7 ビットなのか。

**A** : Z80 CPU が開発された当時は、1K ビットの DRAM がやっと(?)世に出たような時代で、現在のような 4M や 16M ビットの DRAM が出てくるなどというのは想像のはるか彼方の世界でした。こんな時代でしたから、DRAM のリフレッシュ・アドレスは 7 ビットもあれば十分と考えたのだと想像されます。ちなみに、R レジスタが自動的に更新される部分は 7 ビットですが、レジスタ自体は 8 ビットの幅があります。下位 7 ビットの部分は自動的にインクリメントされますが、最上位ビットは書き込んだデータがそのままの状態に残ります。

また現在では Z80 に DRAM を使うことはまずありません。よって R レジスタの値を書き換えても動作に支障はありません。そこでもう一つおもしろい(?)応用例があります。R レジスタはリフレッシュ・サイクルを実行するたびに +1 されるので、M1 サイクルのカウンタともいえます。つまり何命令実行したかの命令カウンタとしても使えるわけです。

**Q** : IXL, IYH などの命令はセカンド・ソースの Z80 でも使える公然の未定義命令になっている。未定義にしたのは何か意味があるのか(IX, IY はあくまでアドレス・ポインタ用として使ってほしいからか?)。

**A** : Z80 CPU は、前述のようにトランジスタの数を減らすべく、マイクロ・コードを使用せず、ハードワイヤ・ロジックで組まれました。このため、設計仕様でない命令が偶然できてしまいました。このようにしてできた命令群、例えば IX, IY レジスタの半分、8 ビットを操作する命令などは、その生い立ちからも未定義になっており、また出荷時にテストも行っておりません。

なお、Z80CPU と命令の上位互換性を持つ CPU、例えば Z380MPU では、これらの Z80CPU での未定義命令群は公式にサポートされています。

**Q** : 掛け算、割り算命令がない。

**A** : まず掛け算や割り算を使用しなければならないアプリケーションというものは(特に制御ソフトウェアでは)あまりないということと、Z80CPU が開発された当時、掛け算や割り算をインプリメントするだけのトランジスタの贅沢ができなかったということによると思います。

〈信垣育司〉



## 〈リスト5〉ビット操作命令の例

SET	1, A	:Aレジスタのビット1をセット
SET	7, (IX+0)	:IXレジスタのメモリのビット7をセット
RES	3, E	:Eレジスタのビット3をリセット
RES	5, (HL)	:HLレジスタのメモリのビット5をリセット
BIT	2, H	:Hレジスタのビット2をチェック
BIT	4, (IY-100)	:IY-100のアドレスのメモリのビット4をチェック

### ▶ OUT 命令

IN 命令とは逆に指定された I/O アドレスに A レジスタの値を出力する命令です。これも同様に、C レジスタによる間接アドレッシングによって 8 ビット汎用レジスタの値を出力することも可能です。

### ▶ ブロック入出力命令

メモリ-メモリ間ブロック転送に LDIR や LDDR 命令があったように、Z80 では I/O 空間とメモリ空間に対しても、ブロック入出力命令があります。

**INIR 命令**は、C レジスタで示される I/O からデータを入力し、HL レジスタで示されるメモリに格納し、これを B レジスタのバイト数だけ繰り返します。1 バイト入力するごとに HL レジスタは+1 され、B レジスタは-1 されます。当然のことながら C レジスタは変化しません。

逆に **OTIR 命令**は、HL レジスタで示されるメモリからデータを読み出し、C レジスタで示される I/O に出力し、B レジスタのバイト数だけ繰り返します。これも 1 バイト出力ごとに HL レジスタは+1、B レジスタは-1 されます。

### ● ビット操作命令(リスト5)

8 ビット中の任意のビットに対するビット操作です。すべての汎用レジスタのすべてのビットに対して可能です。

#### ▶ SET 命令

オペランドで指定されたレジスタのビットをセット(1 にする)します。フラグは変化しません。

#### ▶ RES 命令

オペランドで指定されたレジスタのビットをリセット(0 にする)します。フラグは変化しません。

#### ▶ BIT 命令

オペランドで指定されたレジスタのビットの状態をチェックします。“1” ならゼロ・フラグが“0”、“0” ならゼロ・フラグが“1”です。反転することに注意してください。

### ● ロータイト/シフト命令(リスト6)

8 ビット・レジスタの内容をローテイト/シフトする命令です。

ローテイト命令は、シフトされてはみ出たビット・データが戻って入力ビットとなります。シフト命令は、はみ出たビットはキャリ・フラグに入り、シフト入力

## 〈リスト6〉ローテイト/シフト命令の例

RLC	B	:Bレジスタを左にローテイトする
RL	(IX+0)	:IXレジスタのメモリの内容を左にローテイトする
RRC	C	:Cレジスタを右にローテイトする
RR	(IY+0)	:IYレジスタのメモリの内容を右にローテイトする
SLA	A	:Aレジスタを左にシフトする
SRL	B	:Bレジスタを右にシフトする

## 〈リスト7〉CPU 制御命令の例

NOP		:何もしない命令
HALT		:CPU動作停止命令
IM	2	:割り込みモード2に設定
DI		:/INT割り込み受け付け禁止命令
EI		:/INT割り込み受け付け許可命令

には“0”が入ります。

### ▶ RLC/RL 命令

左へローテイトする命令です。RLC 命令は 8 ビット内で、RL 命令はキャリ・フラグも含めた 9 ビット内でシフトします。どちらもキャリ・フラグにはオペランドの最上位ビットが入ります。

### ▶ RRC/RR 命令

右へローテイトする命令です。RRC 命令は 8 ビット内で、RR 命令はキャリ・フラグも含めた 9 ビット内でシフトします。どちらもキャリ・フラグにはオペランドの最下位ビットが入ります。

### ▶ SLA/SRL 命令

SLA 命令はオペランドの内容を左にシフトし、最上位ビットをキャリ・フラグに入れます。

SRL 命令はオペランドの内容を右にシフトし、最下位ビットをキャリ・フラグに入れます。

### ● CPU 制御命令(リスト7)

#### ▶ NOP 命令

何もしない命令です。無駄な命令のように思えますが、制御系では微妙なタイミングを取ったりするときに使用されます。

#### ▶ HALT 命令

CPU を停止させる命令です。割り込みが入るまで、CPU は動作を停止します。

#### ▶ IM/DI/EI 命令

割り込み関連の命令です。割り込みについては第 6 章を参照してください。

最後に、Z80 の命令を整理してまとめたものを表 4 に示します。

### ◆引用文献◆

- (1) 神崎康宏；トランジスタ技術 SPECIAL No.6, Z80 ハード&ソフトのすべて、p.24.

(トランジスタ技術 1994 年 6 月号に加筆、修正)



表 4<sup>(1)</sup> Z80 の命令一覧 ( ) : オペランドのない命令

●転送命令		●ロード命令		●ビット操作命令	
ロード命令	LD	レジスタ、メモリ間のデータ転送 (8ビット, 16ビット)	ローデータ	シフト	ローデータ
	PUSH	レジスタの内容をスタックへプッシュする (16ビット)			
	POP	スタックの内容をレジスタへポップする (16ビット)			
交換命令	EX	レジスタ・ペア同士の内容をそれぞれ交換する	ローデータ	シフト	ローデータ
	EXX	レジスタ・ペア7BC, DE, HLの内容をBC', DE', HL'の内容と交換する			
	プロック転送命令				
●演算命令	LDI	(DE) ← (HL), DE ← DE + 1, HL ← HL + 1, BC ← BC - 1	ローデータ	シフト	ローデータ
	LDIR	(DE) ← (HL), DE ← DE + 1, HL ← HL + 1, BC ← BC - 1, BC = 0まで繰り返し			
	LDD	(DE) ← (HL), DE ← DE - 1, HL ← HL - 1, BC ← BC - 1			
●出力命令	LDDR	(DE) ← (HL), DE ← DE - 1, HL ← HL - 1, BC ← BC - 1, BC = 0まで繰り返し	ローデータ	シフト	ローデータ
	CPI	A ← (HL), HL ← HL + 1, BC ← BC - 1			
	CPTR	A ← (HL), HL ← HL + 1, BC ← BC - 1, A = 0またはBC = 0まで繰り返し			
●演算命令	CPD	A ← (HL), HL ← HL - 1, BC ← BC - 1	ローデータ	シフト	ローデータ
	CPDR	A ← (HL), HL ← HL - 1, BC ← BC - 1, A = 0またはBC = 0まで繰り返し			
	加減算				
●演算命令	ADD	キャリを含まない加算 (8ビット, 16ビット)	ローデータ	シフト	ローデータ
	ADC	キャリを含む加算 (8ビット, 16ビット)			
	SUB	キャリを含まない減算 (8ビット)			
●演算命令	SBC	キャリを含む減算 (8ビット, 16ビット)	ローデータ	シフト	ローデータ
	CP	比較 (フラグのみ変化する) (8ビット)			
	INC	+1する (8ビット, 16ビット)			
●演算命令	DEC	-1する (8ビット, 16ビット)	ローデータ	シフト	ローデータ
	論理演算				
	AND	アキュムレータとの論理積をとる (8ビット)			
●出力命令	OR	アキュムレータとの論理和をとる (8ビット)	ローデータ	シフト	ローデータ
	XOR	アキュムレータとの排他的論理和をとる (8ビット)			
	汎用算術演算				
●出力命令	DAA	アキュムレータの10進補正	ローデータ	シフト	ローデータ
	CPL	アキュムレータの内容の1の補数をとる			
	NEG	アキュムレータの内容の2の補数をとる			
●出力命令	CCF	キャリ・フラグのビットを反転する	ローデータ	シフト	ローデータ
	SCF	キャリ・フラグをセットする			
	●CPU制御命令				
●出力命令	NOP	プログラム・カウンタを1進める	ローデータ	シフト	ローデータ
	HALT	CPUを停止させる			
	DI	INT割り込みを無効にする (IFF <sub>1</sub> , IFF <sub>2</sub> をリセットする)			
●出力命令	EI	INT割り込みを有効にする (IFF <sub>1</sub> , IFF <sub>2</sub> をセットする)	ローデータ	シフト	ローデータ
	IM	割り込みモードを0~2にセットする			
	●出力命令				

## アクセス・タイミングの計算からバンク切り替えまで

### メモリ周辺回路の設計

村田浩義/北川信孝

#### メモリとは何だろう

##### ● 高速大容量化の進むメモリ

半導体技術の進歩でメモリの大容量化が進み、パソコンに数十 M バイトのメモリを積むことも珍しくなくなってきました。

いまや Z80 マイコンの世界でも、メモリ容量 32 K バイトはかなり小さいほうです。Z80 マイコン・システムなら 256 K ビット・メモリを 2 個つんで、64 K バイト・フル・メモリ・システムを構成できます。

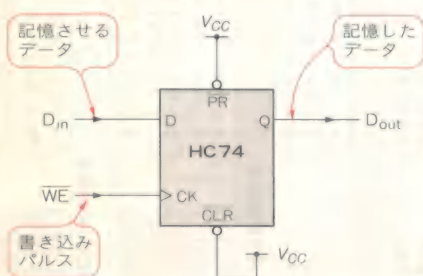
メモリの高速化、大容量化は日進月歩ですが、逆に Z80 などで多用されていた 64 K ビットのメモリは入手しづらくなってきています。このままでは 256 K ビットのメモリも心配です。M バイト時代になっても、Z80 のような組み込み機器用のために、せめて 256 K ビットのメモリだけは作り続けていただきたいものです。

ここではまずメモリとは何か、どんなものがあるか見てみましょう。

##### ● メモリとは

メモリは“1”か“0”かという情報を記憶する素子です。汎用ロジック IC の中では D フリップフロップ (HC74) もメモリとして考えることができます (図 1)。D フリップフロップを複数個並べて、アドレス・バスによって一つだけを選択する回路と、データ・バスにデータを入出力する回路、そして読み出しか書き込み

〈図 1〉 D フリップフロップを使った 1 ビットの記憶



かのコントロール回路を設ければ、りっぱにメモリを構成できます。

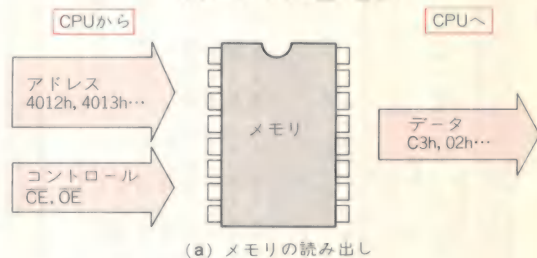
このように、アドレスを指定してデータを読み書きできる素子がメモリです (図 2)。しかし D フリップフロップを並べていたのでは効率が悪いので、**メモリ素子として特化した IC** が作られています。マイコン・システムでは、特に断らない限り半導体メモリのことを略してメモリと呼んでいます。

メモリを大きく分けると、**揮発性メモリ**と**不揮発性メモリ**に分類できます。

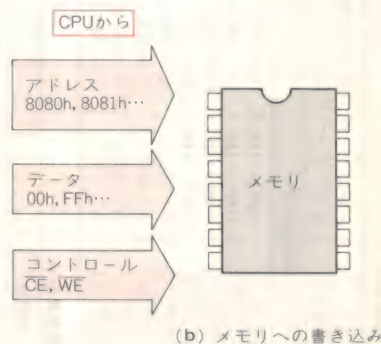
揮発性メモリは、**DRAM** (Dynamic RAM) や **SRAM** (Static RAM) に代表されるような、電源を切るとデータが失われてしまうメモリのことで、読み書きやランダム・アクセスが自在にできるので、**RAM** (Random Access Memory) とも呼びます。

不揮発性メモリは、**UV EPROM** (UltraViolet Eraseable Programmable ROM) に代表される、電源を切ってもデータが失われないメモリです。基本的に

〈図 2〉 メモリの読み書き



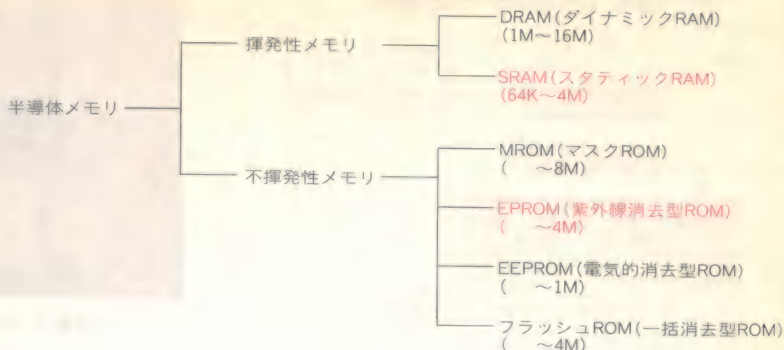
(a) メモリの読み出し



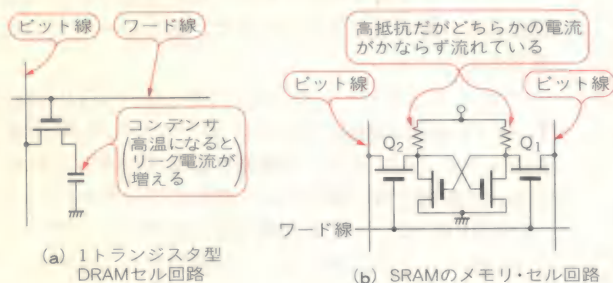
(b) メモリへの書き込み



〈表 1〉  
メモリの分類



〈図 3〉 SRAM と DRAM のデータ記憶構造



〈写真 1〉 SRAM と DRAM の外観 (左から 64 K ビット SRAM, 8 K ビット SRAM, 1 M ビット DRAM)

書き込みができず読み出し専用なので、ROM (Read Only Memory) とも呼びます。

表 1 にメモリの分類を示します。

### ● 揮発性メモリのいろいろ

電源を切ると内容が失われるわけですから、電源 ON ですぐに動作させる組み込みマイコンでは、RAM にプログラムを記憶させておくことはできません。RAM はデータ領域やスタック・エリアとして使われます。

揮発性メモリはさらに DRAM と SRAM に分けられます。図 3 に DRAM と SRAM の内部構造図を、写真 1 に外観を示します。

#### ▶ DRAM とは

DRAM は“0”、“1”の記憶部分がコンデンサにより形成されており、このコンデンサに電荷を充電することで“0”、“1”の情報を記憶します。記憶部分がコンデンサのため、リーク電流により電荷が減少してしまい、ほおっておくと内容が消えてしまいます。

このため、一定時間ごとに一度内容を読み出し再書き込みをします。この作業をリフレッシュと呼びます。またこのような構造のため、高温ではアクセス・タイムも遅くなり、リーク電流も増加します。

また、例えば 1M ビット DRAM はアドレス・ピンが 10 本と、1 M のアドレスを指定するのに半分の本数しかありません。これはアドレスを時分割して指定することでピン数を減らし、実装面積を小さくできる

ようになっているからです。

この時分割アドレス指定のため、 $\overline{\text{RAS}}$  と  $\overline{\text{CAS}}$  によるアドレス制御が必要で、なかなか使いにくいメモリです。

DRAM はパソコンなどでは主メモリとして使われていますが、8 ビット程度のマイコン・システムでは現在ではあまり使われません。

#### ▶ SRAM とは

SRAM はまさに、D フリップフロップを並べたメモリと考えることができます。フリップフロップにより“0”、“1”を記憶しているため、DRAM のリフレッシュのような処理は必要ありません。

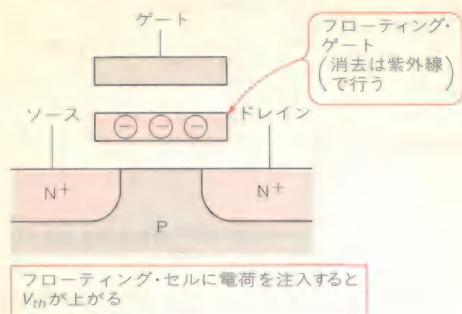
また  $\overline{\text{RAS}}$  や  $\overline{\text{CAS}}$  のような時分割アドレス指定ではなく、そのままアドレスを指定できるので非常に使いやすいメモリです。

データ・バスも 8 本のものが主流で、8 ビット CPU である Z80 などに最適です。またサイクル・タイムとアクセス・タイムが同じなので、高速クロック回路でも設計しやすく、使いやすいメモリです。

ただし DRAM と同様に、高温でリーク電流が増加するので、バッテリー・バックアップする場合は使用温度に留意してバッテリーを決める必要があります。

逆に、低温時に MOS IC は消費電流が増加するの

〈図4〉 EPROM のデータ記憶構造



で、バッテリー動作システムはバッテリーも減りやすくなるので要注意です。

### ● Z80 で使う RAM

以上より、DRAM は M バイト単位でメモリ容量が欲しいときに最適で、Z80 程度のシステムではもはや使われません。Z80 で使う RAM としては SRAM を使うのが一般的です。

### ● 不揮発性メモリのいろいろ

不揮発性メモリは、電源を切っても内容が消えないので、電源 ON で動作する組み込み機器のためのプログラムや初期データ用のメモリとして使われます。

不揮発性メモリには IC 製造時に内容が決まってしまうマスク ROM と、ユーザー側でも専用回路によりデータを書き込むことのできる EPROM とに分けられます。

さらに EPROM にはデータの消去方法の違いにより、紫外線で消去が可能な UV EPROM、電氣的に消去が可能な EEPROM、そして最近使われるようになってきた EEPROM の親戚でもある一括消去型のフラッシュ ROM に分けられます。

図4に EPROM の構造図を、写真2に EPROM の外観を示します。

#### ▶ マスク ROM とは

これは IC 製造時にマスク (IC を作る時のフィルム) で内容が決まってしまうメモリで、大量生産品にしか使いません。また内容の変更もできないので開発時に使うこともありません。

#### ▶ EPROM とは

このメモリに内容を書き込むには、専用の書き込み器 (ROM ライタ) が必要です。この書き込み器で、フローティング・ゲートに電荷を注入することによりデータを書き込みます。

EPROM の中でも開発時などにいちばん使われるのが UV EPROM です。UV EPROM はセラミック・パッケージで、紫外線を照射するための窓がパッケージの上部に開いていてチップが見える構造になっています。紫外線を一定時間照射することによりデータを



〈写真2〉 EPROM の外観

(上: ワritable PROM, 下: UV EPROM)

消去 (フローティング・ゲートの電荷を抜く) できます。

プログラムを書き込んで製品に組み込むときは、太陽光の紫外線が当たらないように、窓をシールなどで塞ぎます。

プラスチック・パッケージで窓のない OTP (One Time Programmable) もあり、窓がないため紫外線を当てることができず、一度書き込んだら消去できません。比較的少量の製品によく使われています。

UV EPROM は一定時間紫外線に当てることでデータを消去するため、多少なりとも時間がかかります。EEPROM は電氣的にデータを消去するので、消去して書き込みを終えるまでの時間が少なくて済みます。フラッシュ ROM に至っては瞬間的に全データを消去できるようになっています。

EPROM は、アドレスの指定が SRAM と同じで、データ・バス幅も 8 ビットと、Z80 と簡単に接続することができる使いやすいメモリです。

### ● Z80 に使う ROM

大量生産するものならまだしも、少量生産品ではマスク ROM は論外です。やはり書き込み器などツールがそろっている UV EPROM が、Z80 システムには最適でしょう。

### ● メモリのアクセス手順

ここではメモリからデータを読み出したり、データを書き込むときの制御の手順を説明します。特に Z80 システム用に最適な SRAM と EPROM について取り上げます。

メモリのアクセスに必要な信号は、アドレス、チップ・セレクト (CE)、そして読み出しか書き込みかの指定である OE や WE です。

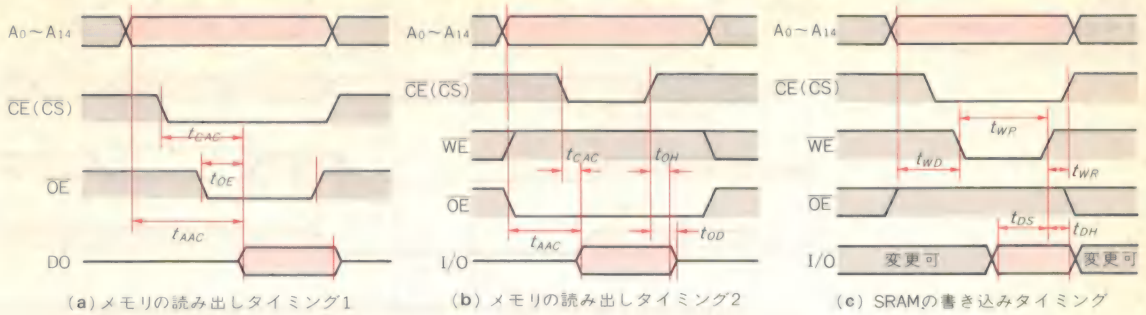
図5に一般的なメモリのアクセス方法と、表2に代表的な 256 K ビット・メモリのアクセス・タイムを示します。

アクセス・タイムとは、アドレスが確定してからデータが正しく出力されるまでの時間をいいます。

高速なクロックで動作するマイコンになればなるほど、このアクセス・タイムが速い必要があります。



〈図5〉 EPROM と SRAM のアクセスのようす



〈表2〉  
代表的な 256 K ビット・  
メモリのアクセス・タイム

型 名	スイッチング特性								メーカ
	$t_{AAC}$ max (ns)	$T_{CAC}$ max (ns)	$t_{OH}$ max (ns)	$t_{WP}$ min (ns)	$t_{DS}$ min (ns)	$t_{DH}$ min (ns)	$t_{WD}$ min (ns)	$t_{WR}$ min (ns)	
MB84256A-10	100	100	40	60	40	0	0	5	富士通
MB84256A-70	70	70	35	50	25	0	0	5	
MB84257-12	120	120		70	45	0	0	5	
HM62256P-12	120		60	90	50	0	0		日立
HM62256P-8	85		45	70	40	0	0		
HM62832HLP/JP-25	25	25	12	15	12	0	0	0	
$\mu$ PD43256G/GU-10L	100	100	50	70	40	0	0	5	NEC
$\mu$ PD43256G/GU-15L	150	150	70	90	60	0	0	5	
$\mu$ PD43258ACR/LA-25	25	25	12	15	12	0	0	0	
TC55257APL-85	85	85	45	60	40	0	0	5	東 芝
TC55257APL-10	100	100	50	70	40	0	0	5	
TC55257APL-12	120	120	60	80	50	0	0	5	

(a) SRAM のアクセス・タイム

まずはアドレスを確定することが先決です。そしてチップ・セレクト(CE)をアクティブにします。そして読み出しにはOEをアクティブにしてデータ・バスにデータを出力させます。ROMもRAMもこれは同じです。RAMの場合はデータの読み出し時はWEをHレベルにしておきます。

また書き込みではWEをアクティブにし、データ・バスに書き込むデータを出力します。そしてWEクロックの立ち上がりでRAMはデータを取り込みます。このときは読み出し信号であるOEをHレベルにしておきます。

これらの制御線は必要のないときはHレベルにしておきます。

#### ● CE や OE の制御タイミングのいろいろ

ここで、図5(a)、(b)のメモリの読み出しタイミングをみてください。この違いは、CEをアクティブにしてからOEをアクティブにしているか、先にOEをアクティブにしてからCEをアクティブにしているかの違いです。基本的にメモリの読み出しはどちらでも動作しますが、一般的には図(a)の方法でアクセスします。

また変則的な例として、ROMのアクセス速度をギ

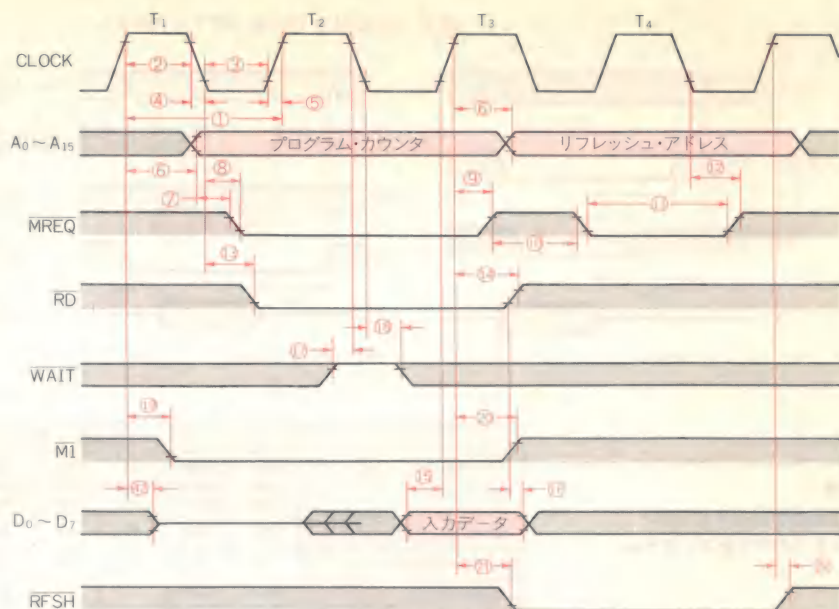
型 名	スイッチング特性			メーカ
	$t_{AAC}$ max (ns)	$t_{CAC}$ max (ns)	$t_{OE}$ max (ns)	
MBM27C256A-20W	200	200	70	富士通
MBM27C256H-10	100	100	45	
MBM27C256H-12	120	120	50	
HN27C256AG-10	100	100	60	日立
HN27C256AG-15	150	150	70	
HN27C256HG-70	70	70	40	
$\mu$ PD27C256AK-12	120	120		NEC
$\mu$ PD27C256AK-15	150	150		
$\mu$ PD27C256AK-20	200	200		
TC57256AD-12	120	120	60	東 芝
TC57256AD-15	150	150	70	

(b) UV EPROM のアクセス・タイム

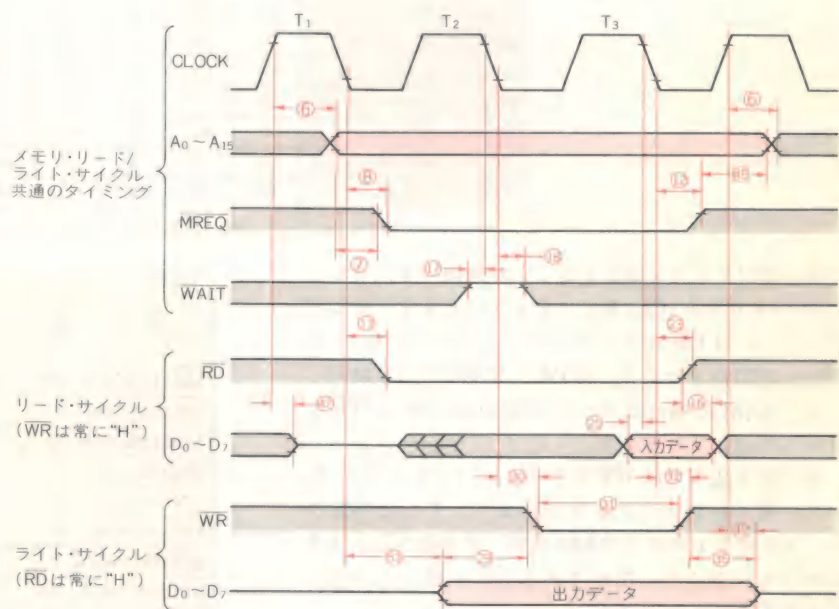
リギリまで使うため、CEをグラウンドに落としたまま、OEだけで読み出し制御をする場合もあります。表2をみるとわかりますが、アドレスが確定してからのアクセス時間 $t_{AAC}$ と、CEがイネーブルになってからのアクセス時間 $t_{CAC}$ は同じです。

ということは、アドレス・デコード回路からの出力をCEに接続した場合、アドレス・デコードの遅延時

〈図 6〉  
オペコード・フェッチ・  
サイクルのタイミング



〈図 7〉  
メモリ・リード/ライト・  
サイクルのタイミング



間だけ、 $t_{AAC}$ によるアクセス時間より遅くなるわけ  
です。

そこで、ROM の場合は読み出し動作だけなわけ  
ですから、アドレス・デコードにより読み出すアドレス  
が決まったら、これを  $\overline{OE}$  に接続してアクセス時間  
をかせる方法があるのです。 $\overline{OE}$  がアクティブになっ  
てからのアクセス時間  $t_{OE}$  は、 $t_{AAC}$  や  $t_{CAC}$  の半分程度と早  
いので、 $\overline{CE}$  をグラウンドに接続しておけば、アドレ  
ス・バスが確定した時点でアクセスが始まり、アドレ  
ス・デコーダでの遅延時間分をあまり気にする必要が

なくなります。

RAM については、書き込み動作があるため、この  
方法を使うのは少々大変です。

## Z 80 のメモリ・アクセス・タイミング

### ● メモリ・アクセス・タイミングの重要性

マイコンは命令をメモリから読み込み、動作します。  
またその命令によって、メモリからデータを読み出し  
たり、メモリにデータを書き込んだりします。



〈表3〉 オペコード・フェッチとメモリ・リード/ライト・サイクル(単位 ns)

番号	記 号	項 目	6MHz 版		8MHz 版		10MHz 版		20MHz 版	
			最小	最大	最小	最大	最小	最大	最小	最大
1	$t_{CC}$	クロック周期	162	DC	125	DC	100	DC	50	DC
2	$t_{wCh}$	クロック“H”パルス幅	65	DC	55	DC	40	DC	20	DC
3	$t_{wCl}$	クロック“L”パルス幅	65	DC	55	DC	40	DC	20	DC
4	$t_{fC}$	クロック立ち下がり時間		20		10		10		10
5	$t_{rC}$	クロック立ち上がり時間		20		10		10		10
6	$t_{dCr(A)}$	クロック立ち上がりからの有効アドレス出力遅延		90		80		65		57
7	$t_{dA(MREQ)}$	MREQ に先立つアドレス出力確定時間	35		20		5		-15	
8	$t_{dCr(MREQ)}$	クロック立ち下がりから MREQ = “L” になるまでの遅延		70		60		55		40
9	$t_{dCr(MREQ)}$	クロック立ち下がりから MREQ = “H” になるまでの遅延		70		60		55		40
10	$t_{wMREQH}$	MREQ “H” パルス幅	65		45		30		10	
11	$t_{wMREQl}$	MREQ “L” パルス幅	132		100		75		25	
12	$t_{dCr(MREQr)}$	クロック立ち下がりから MREQ = “H” になるまでの遅延		70		60		55		40
13	$t_{dCr(RDf)}$	クロック立ち下がりから RD = “L” になるまでの遅延		80		70		65		40
14	$t_{dCr(RDf)}$	クロック立ち上がりから RD = “H” になるまでの遅延		70		60		55		40
15	$t_{SD(Cr)}$	クロック立ち上がりに対するデータ・セットアップ時間	30		30		25		12	
16	$t_{hD(RDf)}$	RD 立ち上がりに対するデータ・ホールド時間	0		0		0		0	
17	$t_{SWAIT(Cf)}$	クロック立ち下がりに対する WAIT セットアップ時間	60		50		20		7.5	
18	$t_{hWAIT(Cf)}$	クロック立ち下がり後の WAIT ホールド時間	10		10		10		10	
19	$t_{dCr(Ml)}$	クロック立ち上がりから Ml = “L” になるまでの遅延		80		70		65		45
20	$t_{dCr(Ml)}$	クロック立ち上がりから Ml = “H” になるまでの遅延		80		70		65		45
21	$t_{dCr(RFSHf)}$	クロック立ち上がりから RFSH = “L” になるまでの遅延		110		95		80		60
22	$t_{dCr(RFSHf)}$	クロック立ち上がりから RFSH = “H” になるまでの遅延		100		85		80		60
23	$t_{dCr(RDf)}$	クロック立ち下がりから RD = “H” になるまでの遅延		70		60		55		40
25	$t_{SD(Cf)}$	クロック立ち下がりに対するデータ・セットアップ時間 (M2, M3, M4, M5 サイクル時)	40		30		25		12	
29	$t_{dD(WRf)}$	WR 立ち下がりから先立つデータ確定時間	22		5		40		-10	
30	$t_{dCr(WRf)}$	クロック立ち下がりから WR = “L” になるまでの遅延		70		60		55		40
31	$t_{wWR}$	WR パルス幅	132		100		75		25	
32	$t_{dCr(WRf)}$	クロック立ち下がりから WR = “H” になるまでの遅延		70		60		55		40
35	$t_{dWRP(D)}$	WR = “H” になってからの出力データ保持時間	30		15		10		0	
42	$t_{dCr(Dz)}$	クロック立ち上がりからデータ・バス・フロート状態までの遅延		80		70		65		40
45	$t_{dCr(A)}$	MREQ, IORQ, RD または WR からのアドレス保持時間	35		20		20		0	
53	$t_{dCr(D)}$	クロック立ち下がりからデータ出力までの遅延		130		115		110		75

このメモリからの命令、データの読み出しやデータの書き込みには、それぞれ制御の方法やタイミングと  
いうものが決まっています。

この制御の方法やタイミングを守らないと、正しく  
メモリからデータを読み書きすることができません。  
このタイミングを守らないと、どんなデータが読み書き  
されるかわかったものではありません。CPU にし  
てみれば、読み込んだ命令が実は正しくないかもしれ  
ないという状況が起こり得るのです。

ここではまず、Z80 がどのようなタイミングでメモ  
リを読み書きするかを説明します。オペコード・フェ  
ッチ・サイクルやメモリ・リード/ライト・サイクルにつ  
いては第1章で説明しましたが、ここではさらに詳しく、  
図6と図7にそのタイミングを、表3にアクセ  
ス・タイムを示します。

## ● オペコード・フェッチ・サイクル

Z80 が命令を読み込むことを、オペコード・フェッ  
チと呼び、このときの CPU の状態をオペコード・フ  
ェッチ・サイクルと呼びます。

Z80 の命令は、オペコードと呼ばれる動作そのものを  
指定するデータと、オペランドと呼ばれるレジスタや  
アドレスを指定するデータの組み合わせで構成されま  
す。

命令によってバイト数が異なり、Z80 では最短バイ  
ト数の命令は1バイトで、最長バイト数の命令は4バ  
イトで構成されています。

まず初めに CPU は、T<sub>1</sub>の立ち上がりで PC(プログ  
ラム・カウンタ)の内容をアドレス・バスに出力し、Ml  
を立ち下げます。そして T<sub>1</sub>の立ち下がりでは MREQ  
(メモリ要求信号)と RD(読み出し要求)をアクティブ

にします。

これらの信号で選択されたメモリからデータが出力され、CPU は  $T_3$  の立ち上がりでデータを取り込みます。

そして次の  $T_3$ 、 $T_4$  の二つのサイクルで、CPU は DRAM に対するリフレッシュ信号を出力し、内部では先ほど取り込んだデータ(オペコード)の解析(デコード)と実行がなされます。このとき出力されるリフレッシュ信号は、 $\overline{MREQ}$  と  $RFSH$  信号がアクティブになり、リフレッシュ・アドレスがアドレス・バスに出力されます。

### ● オペコード・フェッチ・サイクルをメモリからみると

このようにオペコード・フェッチ・サイクルは、4 ステート(クロック)サイクルなのですが、 $T_3$  と  $T_4$  ではアドレス・バスにリフレッシュ・アドレスが出力されるので、**実質的な命令読み込みのためのアドレス出力は、 $T_1$  と  $T_2$  の時間だけです。**

メモリにしてみれば、アドレスが確定したあと  $T_3$  の立ち上がりで CPU がオペコードを読み込む前に、指定されたアドレスのデータ、つまりこの場合は命令をデータ・バスに出力しなければなりません。これがメモリが守るべきアクセス・タイミングです。

またオペコードが ROM 上に置かれようが RAM 上に置かれようが、CPU はそのことについてまったく感知していないことを頭に入れてください。

たしかに電源投入時はプログラムは ROM 上にしかありませんが、その途中で RAM 上にプログラムを展開して動作するというシステムも構成可能です。

つまり ROM に対しても RAM に対しても、通常は同じタイミングでオペコード・フェッチが行われるわけです。

### ● メモリ・リード/ライト・タイミング

次はオペコードの読み出しではなく、オペランドやデータをメモリから読み込んだり、データを書き込んだりするときのタイミングをみてみます。Z80 ではこれを**メモリ・リード/ライト・サイクル**と呼んでいます。

オペコード・フェッチが  $T_1$  と  $T_2$  の 2 クロック時間であったのに対して、メモリ・リード/ライト・サイクルは  $T_1$  から  $T_3$  の立ち下がりまでの 2.5 クロック時間です。オペランドの読み込み時は PC の内容が、メモリの読み込み時はそのメモリ・アドレスがアドレス・バスに出力されます。このとき、 $\overline{MREQ}$  と  $\overline{RD}$  がアクティブになり、メモリから出力されたデータを CPU は  $T_3$  の立ち下がりで取り込みます。

データの書き込み時は、メモリ・アドレスがアドレス・バスに出力され、 $\overline{MREQ}$  と  $\overline{WR}$  がアクティブになり、CPU からデータ・バスに書き込むデータが出力されます。

### ● メモリ・リード/ライト・サイクルをメモリからみると

オペコード・フェッチ・サイクルでは、 $T_3$  の立ち上がりでデータが CPU に取り込まれていたのが、メモリ・リード・サイクルでは、 $T_3$  の立ち下がりでデータが CPU に取り込まれるので、**0.5 クロック時間アクセスに余裕がある計算**になります。

またメモリ・ライト・サイクルでは、 $\overline{MREQ}$  がアクティブである時間はメモリ・リード・サイクルと同じ 2.5 クロック時間ありますが、 $\overline{WR}$  がアクティブである時間はほぼ 1 クロック分で、 $T_2$  の立ち下がりです。

つまりデータの読み込みに関しては、オペコード・フェッチ・サイクルを満足できるのであれば問題はないが、書き込みに関しては  **$\overline{WR}$  信号の立ち下がりのタイミングとその時間に注目**する必要があります。またメモリへの書き込みのタイミングは  $\overline{WR}$  クロックの立ち上がりで行います。

またオペコード・フェッチ同様、メモリ・リード・サイクルも ROM と RAM の両方に対して考えられ、また同じタイミングが要求されます。しかしメモリ・ライト・サイクルだけは、メモリへの書き込み動作ですから、RAM についてのみ考えればよいわけです。

## Z80 とメモリの接続

### ● マイコン・システムに必要な ROM と RAM

Z80 は電源を入れるとアドレス 0000h 番地から実行を開始します。したがって**電源を切ってもプログラムを記憶しておくことのできる ROM が必ず必要**です。

また簡単な動作だけであれば、データの保存にレジスタだけを使い、RAM をいっさい使用しなくてもシステムを構成することも可能ではあります。

しかし扱うデータが多かったり、CALL 命令を使ったサブルーチン・コールをするとき、また割り込みを使う場合などは、ワーク・エリアやスタック・エリアが必要となるので、RAM は不可欠です。

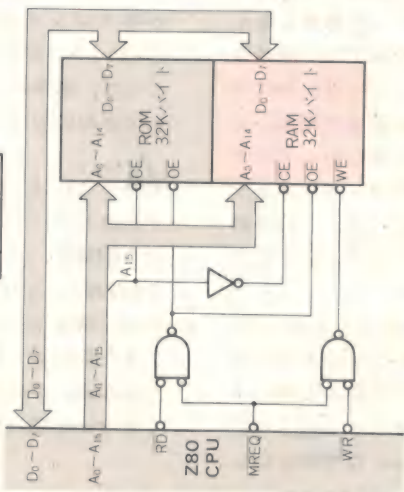
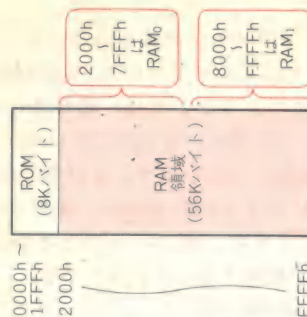
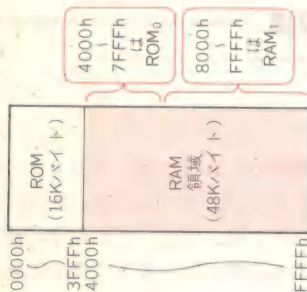
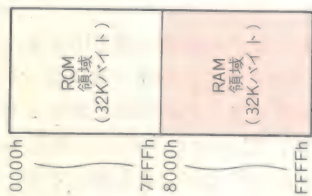
### ● メモリ空間と I/O 空間

Z80 は 8080 系の流れをくむ CPU ですから、メモリ空間と I/O 空間の二つのアドレス空間をもっています。このメモリと I/O のアドレスの指定はアドレス・バスが共用されます。このため、いま CPU がアドレス・バスに出力しているのはメモリのアドレスなのか、I/O のアドレスなのかを選択するために、 $\overline{MREQ}$  と  $\overline{IORQ}$  (I/O 要求信号)があります。

それぞれ、L レベルでアクティブになり、当然両方が同時にアクティブになることはありません。メモリ空間と I/O 空間の選択が両方同時に起こることはありえないので、二つの信号を 8086 のように 1 本で



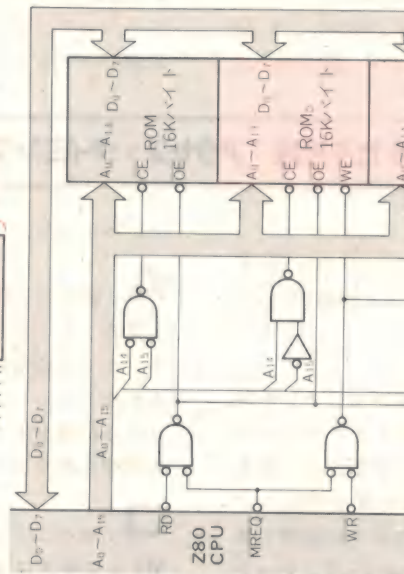
図8 ROMとRAMの分割例



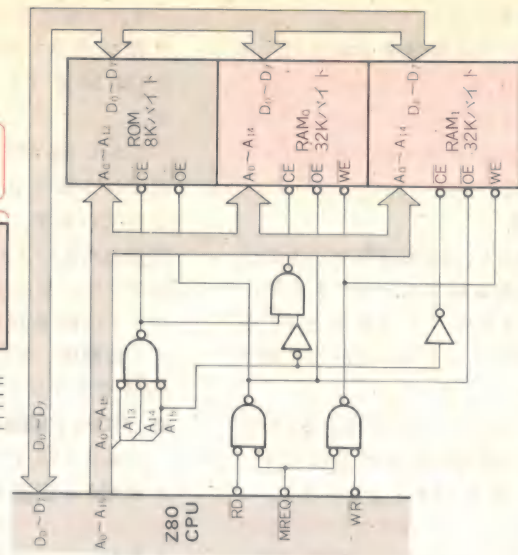
(a) ROM32Kバイト, RAM32Kバイトのメモリマップ



(b) 32Kバイトのメモリを16Kバイトで使用する



(c) ROM16Kバイト, RAM48Kバイトのメモリマップ



(d) ROM8Kバイト, RAM56Kバイトのメモリマップ

済ませている CPU もあります。

いずれにしろ Z80 でメモリを接続する場合は、メモリ選択信号である **MREQ** を、必ずメモリの **CE** または **OE** と **WE** に接続しなければなりません。この接続を忘れると、I/O 空間のアクセスのときにもメモリが選択されてしまいます。

### ● Z80 のメモリ・マップ例

Z80 はアドレスが 16 本あり、64 K ( $2^{16}$ ) バイトのメモリ空間 (0000~0FFFFh) を直接アクセスできます。

電源投入時など、Z80 はリセットされると PC を 0000h にリセットし、命令フェッチは 0000h から始まります。

また現在、入手しやすいメモリとしては、256 K ビットの容量の EPROM と SRAM があげられます。256 K ビット/8 ビットで、32 K バイトとなります。

このことから、前半のアドレス 0000h~7FFFh の 32 K バイトをプログラム書き込み済みの EPROM に、後半のアドレス 8000h~0FFFFh の 32 K バイトを SRAM にしたシステムが多く見られます [図 8(a)]。

またプログラム領域は少なくてもよいけれども、データ領域が欲しいという場合は ROM 8 K バイト、RAM 56 K バイトといった構成 [図 8(c)] も可能です。

しかし 8K バイトのメモリが欲しいといっても、最近では 64K ビット以下のメモリの入手が困難になってきています。このようなときは、図 8(d) のように容量の大きいメモリを使い、余ったアドレス・ビットはグラウンドに接続して使います。この例では、32K バイトの前半 16K バイトのみを使い、後半の 16K バイトは使用していません。8K バイトのときは、 $A_{14}$  と  $A_{13}$  をグラウンドに接続することになります。

### ● デコード回路のいろいろ

メモリといっても 64 K バイトの空間の間に、ROM と RAM が配置されるわけですから、メモリ空間の中でもどのメモリを選択するかは制御が必要です。このような、どのメモリを選択するかは回路をメモリ・アドレスのデコード回路と呼びます。ROM と RAM の容量がいくらでどのアドレス領域に振り分けるかは、

## Z80 七不思議 アクセス・タイミング編

**Q** : オペコード・フェッチ時のリフレッシュ・サイクルは不要。Z80CPU にリフレッシュ回路(というかカウンタ)を入れた理由は?

**A** : Z80CPU では、オペコードをフェッチした後、命令をデコードするのに 1 クロック・サイクルかかります。この間、バスをアイドルにすることも考えられますが、これを Z80CPU では時間を無駄にすることなくリフレッシュ動作を行うように設計しました。今でこそ疑似 SRFM や、セルフ・リフレッシュ機能を内蔵し、とくにリフレッシュを意識しなくても良い DRAM がありますが、Z80CPU が開発された頃のマイコンは、リフレッシュを複雑な外付け回路により行っていました。Z80 の登場当時は、このリフレッシュ・サイクルは非常に使いかたのよいものでした。ちなみに、このオペコード・フェッチ時にリフレッシュ・サイクルを入れる、というのはザイログ社が特許をもっているアイデアです。

**Q** : オペコード・フェッチは  $T_3$  の立ち上がりで、メモリ・リード/ライト・サイクルは  $T_3$  の立ち下がりデータが取り込まれる。メモリから見ると同じメモリ・リードなのにオペコード・フェッチのほうがアクセス・タイムが厳しいのはなぜか、なぜ同じにし

なかったのか。

**A** : 確かにご指摘のとおり、同じメモリ・リードと見ることもできますが、オペコード・フェッチ・サイクルでは命令を早く取り込み、その命令が何であるかをデコードする必要があります。そのため、取り込み位置を半クロック早くして、デコードのための時間を稼ぐように設計したのだと思います。

**Q** : オペコード・フェッチ・サイクル時、なぜ  $\overline{M\bar{I}}$  より半クロック遅れて  $\overline{MREQ}$  が出るのか。  $\overline{M\bar{I}}$  が半クロック早く出るなら、オペコード・フェッチ・サイクル時のメモリ・リードをもう半クロック早くして、通常のメモリ・リードと同じアクセス時間にできたのではないか。

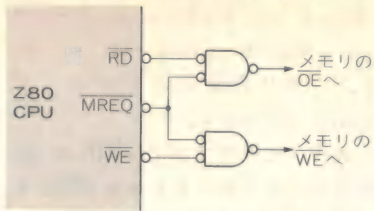
**A** : これは、あくまでも推測の域を出ませんが、オペコード・フェッチ・サイクル時の RD 信号制御を行う回路を、メモリ・リードの回路と共有させるように設計したため、 $T_1$  の立ち下がりで変化させるのが一番楽だったからではないかと思います。

また仮に早くしてもアドレス確定時間は変わりませんから、それほどアクセス・タイムを稼ぐことはできないと思われます。

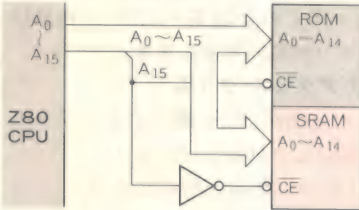
〈信垣育司〉



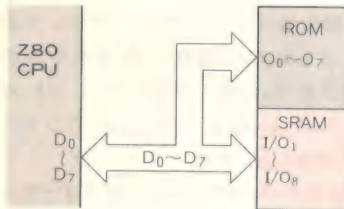
〈図9〉 Z80 とメモリとの接続法



(a) MREQとRD, WEによるメモリ読み書き信号



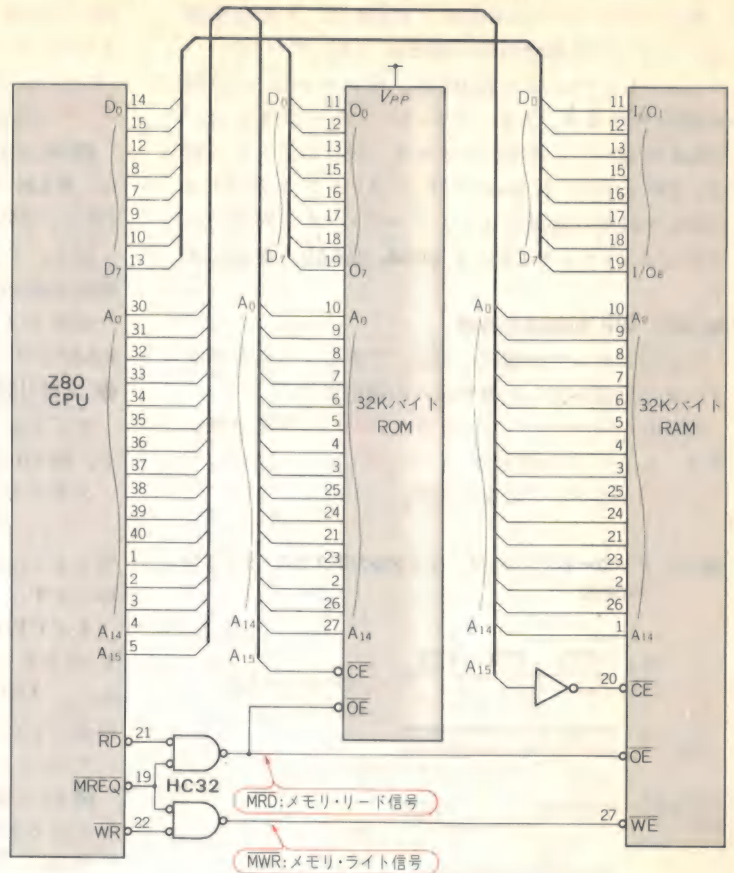
(b) インバータによるデコード回路 (32KROMと32KROM)



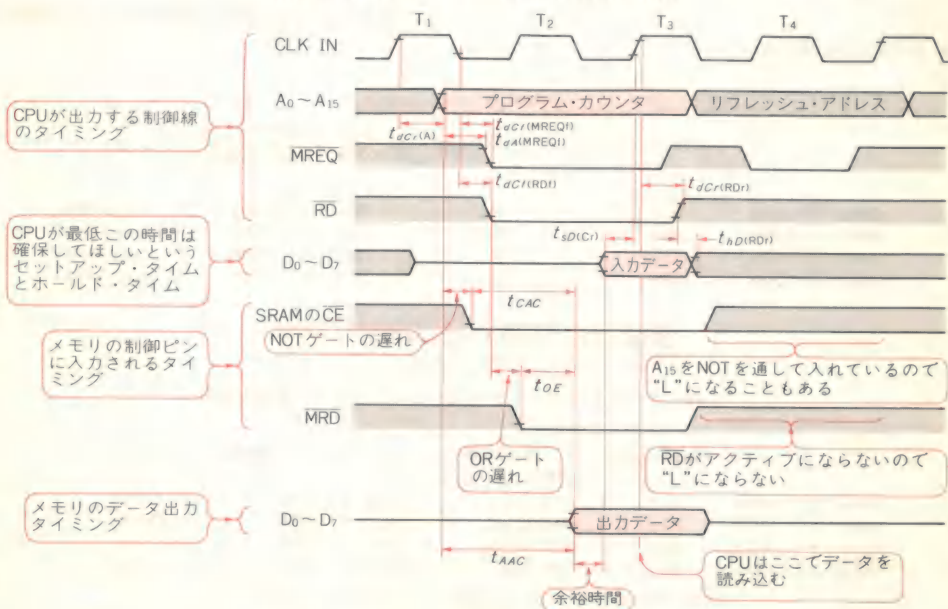
(c) データ・バスの接続

この三つの要素が成り立ってメモリ・アクセスが行われる

〈図10〉 32Kバイト ROM/RAM の回路例



〈図11〉 オペコード・フェッチ・サイクルの動作



このデコード回路で決めます。

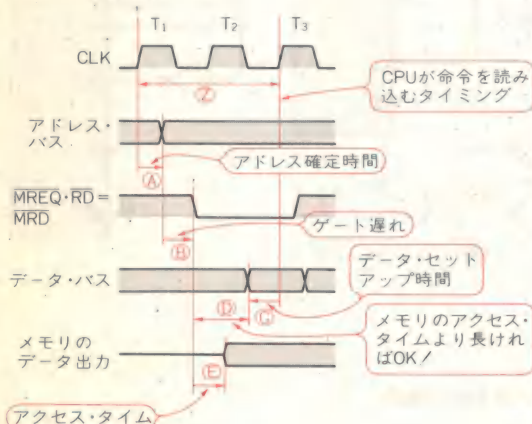
32 K バイトずつの ROM と RAM で、メモリ空間をデコードする場合の例が図 8 (a) です。アドレス・バス  $A_{15}$  が L レベルなら ROM が、H レベルなら RAM が選択されます。 $A_{15}$  にインバータを一つ通せば、RAM のセレクト信号ができます。残りのアドレス線は、Z80 と同じ名前のアドレス・ピンを ROM と RAM それぞれ直結します。データ・バスも同様に、同じビット・ピンを CPU と ROM、RAM に接続します。

### ● RD, WR 制御線の処理

これでアドレスの選択は正しくできるようになります。あとは読み出ししか書き込みの指定です。

Z80 のリード/ライトの制御は、RD と WR で行います。しかしこの信号は、I/O に対してのリード/ライトでもアクティブになります。

〈図 12〉 オペコード・フェッチ・サイクルのアクセス・タイミングの計算



(a) アクセス・タイミングの考え方

ここではメモリに対してのリード/ライト信号が必要なのですから、図のように  $\overline{MREQ}$  信号と負論理 AND を取り、メモリ読み出し信号として  $\overline{MRD}$  (Memory Read) を、メモリ書き込み信号として  $\overline{MWR}$  (Memory Write) を作ります。

ROM は読み出し専用メモリですから  $\overline{MRD}$  を  $\overline{OE}$  に、RAM は読み書きできるメモリですから  $\overline{MRD}$  を  $\overline{OE}$  に、 $\overline{MWR}$  を  $\overline{WE}$  に接続します。

以上、アドレス・バス、データ・バス、各種制御線の接続方法を図 9 にまとめます。また図 10 にもっとも一般的なメモリ・マップである 32 K バイトの ROM/RAM システムのメモリ回路例を示します。

### ● メモリ読み出しのアクセス・タイム

さてここで設計した回路が正しく動作するかどうかを、図 10 の回路を例にして検証してみます。

メモリの読み出しについては、時間的にオペコード・フェッチ・サイクルがいちばん厳しくなることは説明しました。そこでこの時間でアクセス・タイムを計算します。

まず CPU 側で重要な各タイミングは、アドレスが  $T_1$  の立ち上がりからどれだけ遅れて確定するかの  $t_{acr}(A)$ 、 $\overline{MREQ}$  や  $\overline{RD}$  が  $T_2$  の立ち下がりからどれだけ遅れて立ち下がるかの  $t_{dcf}(\overline{MREQ})$  と  $t_{dcf}(\overline{RD})$ 、そしてデータ・セット・アップ時間  $t_{sd}(Cr)$  です。

図 11 に図 10 の回路のオペコード・フェッチ・サイクルのときのアクセスのようすを示します。

問題は、デコード回路で使われているロジック IC によるゲート遅れです。ここでは経験的に平均遅延時間の 1.5 倍で計算することにします。

ROM の  $\overline{CE}$  については  $A_{15}$  が直接接続されているので、アドレスの確定と  $\overline{CE}$  の立ち下がりとは同じと考えられます。また  $\overline{OE}$  を制御している  $\overline{MRD}$  にしても、

	②	③	④	⑤	⑥
2T - アドレス確定時間 - デコード時間 - データ・セットアップ時間 = CPUがデータを 読み込むまでの時間 > アクセス・タイム					
	$t_{acr}(A)$		$t_{sd}(Cr)$		
6MHz = $2 \times 167ns$ -	90ns	-	0ns	-	30ns = 214ns > $t_{AAC}(200ns)$
8MHz = $2 \times 125ns$ -	80ns	-	0ns	-	30ns = 140ns > $t_{AAC}(120ns)$
10MHz = $2 \times 100ns$ -	65ns	-	0ns	-	25ns = 110ns > $t_{AAC}(100ns)$
20MHz = $2 \times 50ns$ -	57ns	-	0ns	-	12ns = 31ns > $t_{AAC}(30ns)$

図 10 の回路では SRAM の  $\overline{CE}$  に NOT ゲートが入るのでそのゲート遅れを入れる

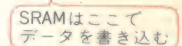
(b) オペコード・フェッチの  $\overline{CE}$  の立ち下がりからの最大時間

	②	③	④	⑤	⑥
1.5T - RD 確定時間 - ゲート遅れ時間 - データ・セットアップ時間 > アクセス・タイム					
	(MRD)		$t_{sd}(Cr)$		
6MHz = $1.5 \times 167ns$ -	80ns	-	9(HC32)ns	-	30ns = 132ns > $t_{OE}(70ns)$
8MHz = $1.5 \times 125ns$ -	70ns	-	9(HC32)ns	-	30ns = 78ns > $t_{OE}(60ns)$
10MHz = $1.5 \times 100ns$ -	65ns	-	6(AC32)ns	-	25ns = 54ns > $t_{OE}(50ns)$
20MHz = $1.5 \times 50ns$ -	40ns	-	6(AC32)ns	-	25ns = 29ns > $t_{OE}(25ns)$

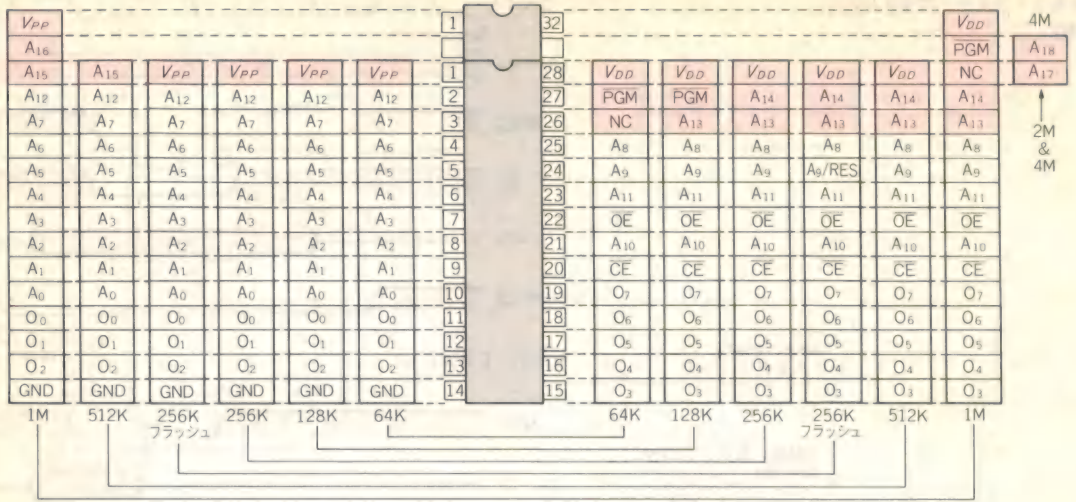
(c) オペコード・フェッチの  $\overline{OE}$  の立ち下がりからの最大時間



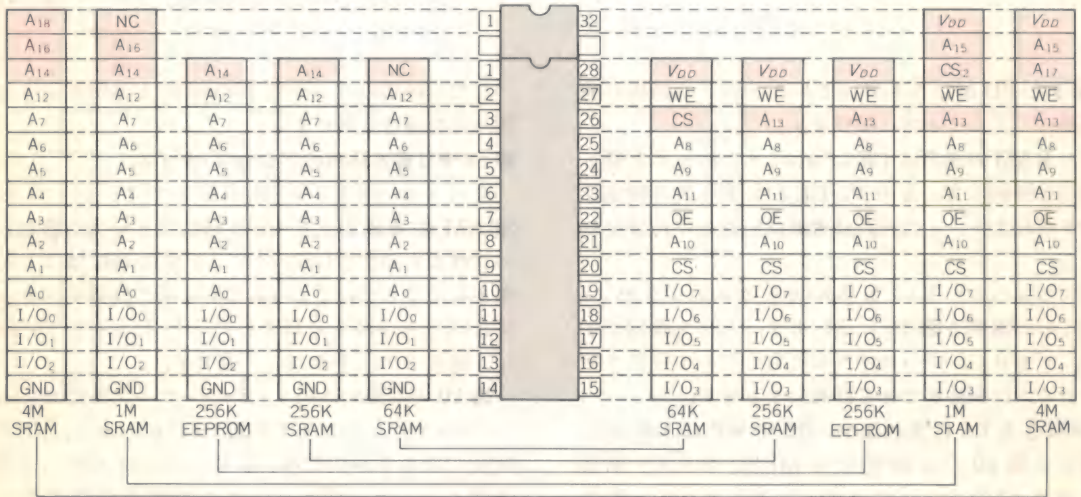
メモリ・ライト・サイクルの動作



〈図 14〉 64 K~4 M ビットの各種メモリのピン配置



(a) UVEPROMとフラッシュROMのピン配置



(b) SRAMとEEPROMのピン配置

当然、さらに複雑なデコード回路や、後述するバックアップ回路のためのゲートを追加すれば、よりアクセス速度の速いメモリが必要になります。

### ● 64 K~256 K ビット・メモリの対応

図 14 に 64 K~4 M ビットの EPROM と SRAM のピン配置図を示します。

256 K ビットのメモリの場合、1 番ピンと 27 番ピン以外のピンは、ROM も RAM も同じピン配置である点を頭に入れておくとよいでしょう。SRAM と EEPROM は、1 番ピンが  $A_{14}$ 、27 番ピンが  $\overline{WE}$  信号です。UV EPROM とフラッシュ ROM は、1 番ピンが  $V_{PP}$ 、27 番ピンが  $A_{14}$  です。

このフラッシュ ROM は UV EPROM とピン配置がまったく同じなので、パターンの変更なしで置き換

えができます。EEPROM を使う場合は、あらかじめ切り替え回路をパターン化しておきます。

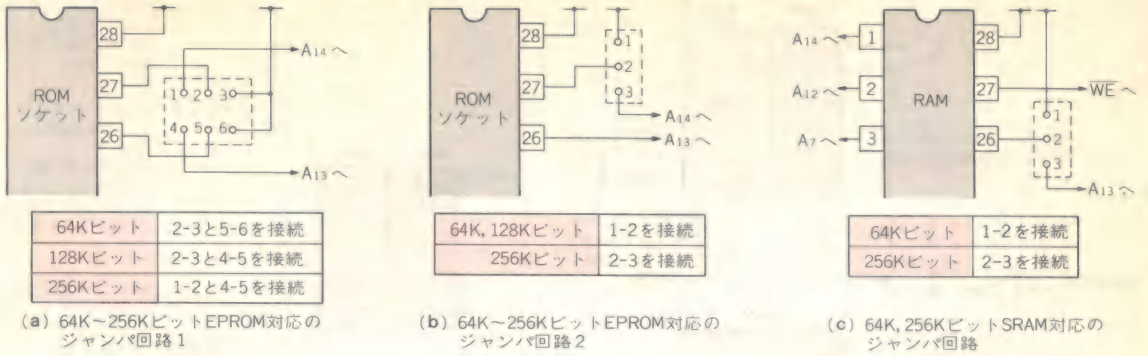
また 64 K~256 K ビットのメモリに対応するためのジャンパ・ピンの回路を図 15 に示します。これで同じ 28 ピン・パッケージであることを利用して、ジャンパ・ピンだけで異なる容量のメモリにも対応させることができます。

またこのピン配置を考えると、64 K ビットや 128 K ビット用の ROM ソケットに 256 K ビットの EPROM を差し込むこともできます。ただし、 $A_{14}$  や  $A_{13}$  にあたるピンの処理で注意する点があります。

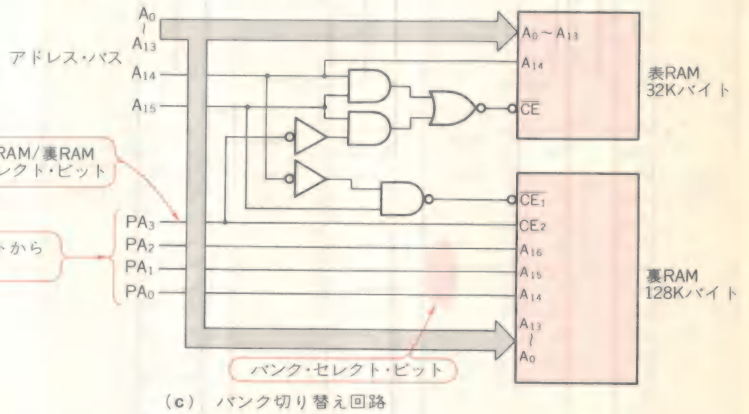
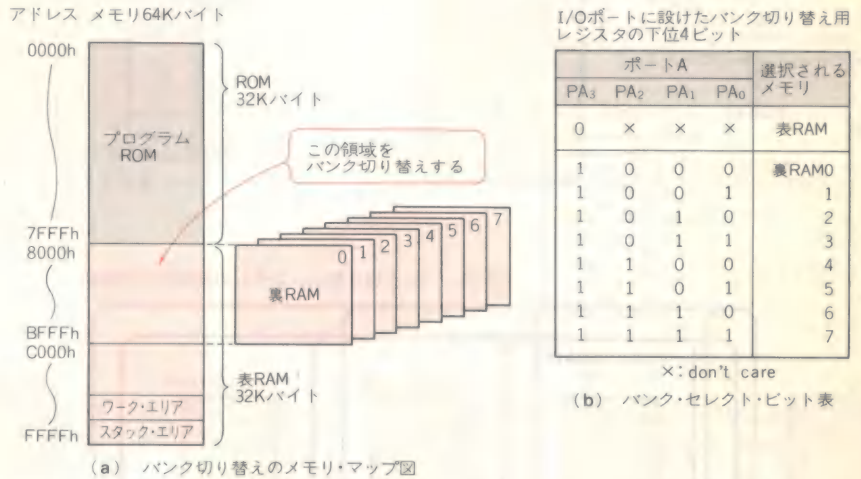
これらのピンの基板上の処理によっては、ROM ライタでの書き込み開始アドレスが、ROM の物理的な 0000h からでなく、6000h や 4000h といった  $A_{14}$  や  $A_{13}$



〈図 15〉 64 K~256 K ビット ROM と RAM 対応ジャンパ回路



〈図 16〉  
バンク切り替えの例



のビットが“1”であるアドレスから書き込まなければならない点です。CPU 側はアドレス 0000h からプログラムを読み込もうとしますが、ROM にしてみれば、A<sub>14</sub>や A<sub>13</sub>のビットが“1”であるアドレスに対してアクセスされることになるからです。

またこのアドレスは、基板の回路が図 15 (a) のようになっているか、図 15 (b) のようになっているかでも違います。図 (a) では A<sub>13</sub> は H レベル固定ですが、図 (b)

では CPU の A<sub>13</sub> が接続されることになります。

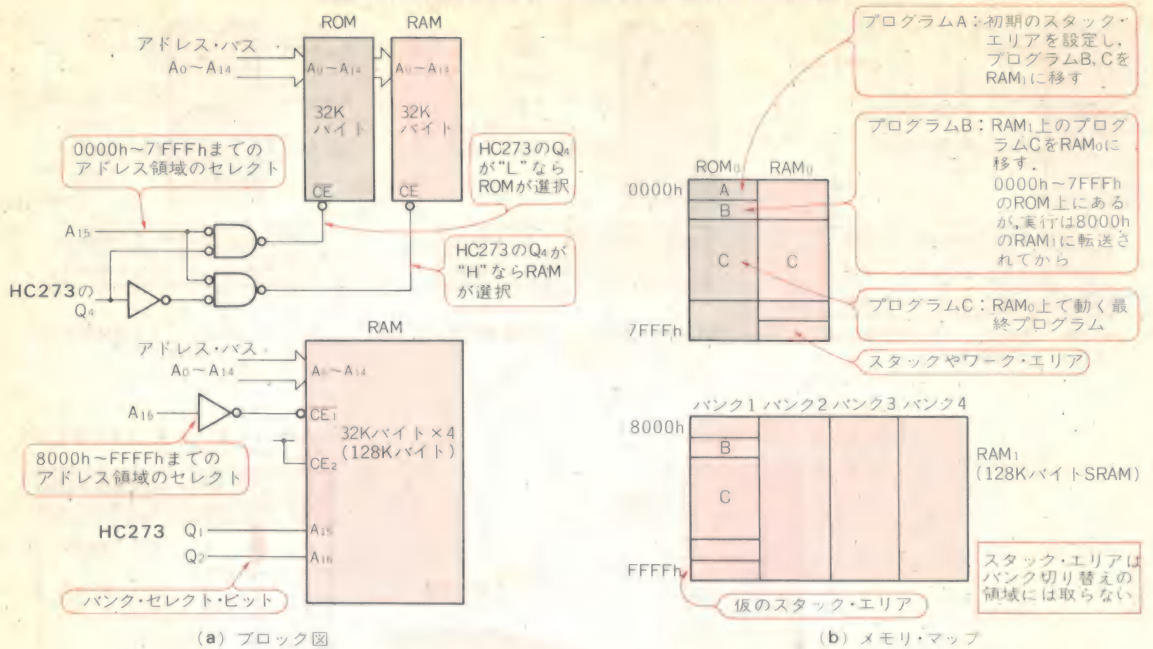
EPROM の V<sub>PP</sub> や PGM などの書き込み制御ピンは、電源に直結します。

## さらに大容量のメモリを使うために

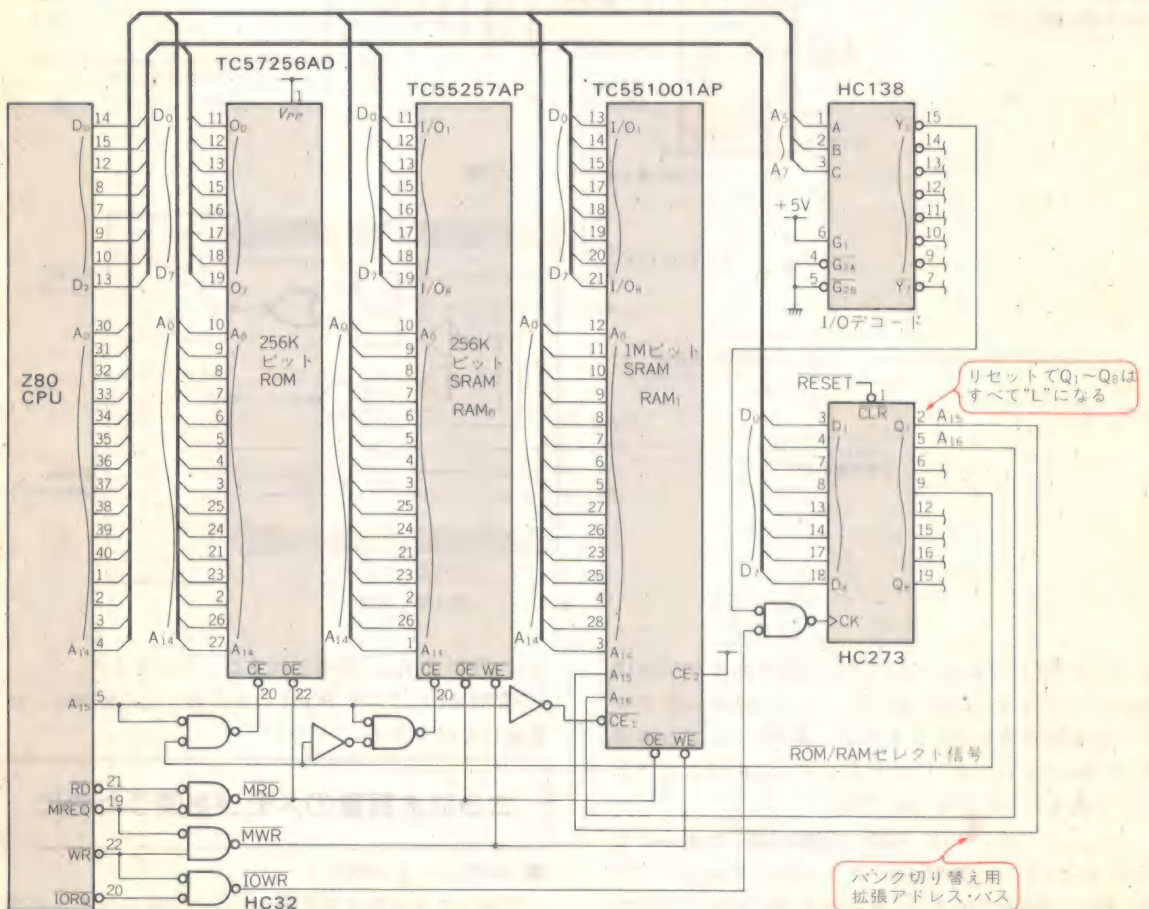
### ● 64 K バイトは狭い！

現在の半導体事情を反映してか、Z80 システムにお

〈図17〉 64 Kバイト・オールRAM 領域にできるメモリ回路



〈図18〉 バンク切り替えシステムの具体的な回路例





いても64Kバイトのメモリ容量に限界が見えてきているのも事実です。最近ではC言語などの高級言語でプログラムを組むことが多く、少しでも凝ったことをやり始めると、64Kバイトではあっという間にメモリを使いきってしまうことがあります。

しかし、Z80には16本のアドレス・バスしかありません。16本ではどう指を折って数えても64Kバイト以上のメモリを数えられません。Z80で64Kバイト以上のメモリを使う方法はないのでしょうか。

そこでどこからか17、18本目のアドレス・バスを無理矢理作り、64Kバイト以上のRAMを管理したり、64Kバイト以内のあるアドレス領域を切り替えて使い、**トータル的に64Kバイト以上のメモリを使う**ようなシステムが考え出されました。このようなシステムを**バンク切り替え**と呼びます。

### ● メモリ・バンク切り替えの具体的な例

バンク切り替えでは、ワーク・エリアとスタック・エリアを切り替わらないアドレス領域に確保しないと、CPUは暴走(プログラムの動作がおかしくなる)してしまいます。また切り替わるアドレス領域にPCがあるときにバンク切り替えしても暴走します。それはそうです。切り替えたとたんプログラム自身がメモリ・バンクの裏へ消えてしまうのですから。

図16にアドレス8000h~0BFFFhにバンク切り替え用の窓を開け、バンク切り替え領域の大きさを16Kバイトとした例を示します。

I/O空間に設けたPIO(パラレルI/O)についての設計法は、次章を参照していただくとして、とりあずここでは単にPIOのポートAの下位4ビットをバンク選択用に使うとだけ説明しておきます。またリセット状態でPA<sub>0</sub>~PA<sub>3</sub>はLレベルになります。

PA<sub>3</sub>がLレベルなら表RAMである256KビットSRAMが、CPUに対してアドレス8000h~0FFFFhに選択されます。Hレベルならアドレス8000h~0BFFFhの領域が裏RAMになります。

さらに裏RAMには1MビットSRAMが使われているので、1M/8ビットで128Kバイトの容量があります。これを16Kバイトのバンクに分けるので、全部で8バンク切り替えが可能になります。この8バンクをPA<sub>2</sub>~PA<sub>0</sub>の3ビットで選択します。

バンクの切り替えはOUT命令で、選択したいバンクの値をPIOのポートAの下位4ビットに出力して設定します。

### ● 64Kバイト・オールRAM

システムによっては、ROM領域までRAMにしたときがあります。アドレスの前半の32Kバイトの表にROM、裏にRAM、後半の32KバイトにRAMのバンクを四つ配置したシステム例を図17に示します。

### ＜リスト1＞ バンク切り替えプログラム例

```
*****
;* Z80 BANK TEST PROGRAM at Mar.20.1994 by H.MURATA *
*****
; *** MEMORY SYMBOL ***
STACK0 EQU 0000H :SP=FFFF+1 START
STACK1 EQU 8000H :SP=7FFF+1 RESTART
; *** I/O SYMBOL ***
IOCE01 EQU 000H :LOW MEMORY SELECT

*****
;* PROGRAM START
*****
ORG 0000H
LD SP, STACK0

LD HL, 0100H :RAM0(アドレス 0100h)から
LD DE, 8100H :RAM1(アドレス 8100h)へ
LD BC, 7700H :7700hバイト
LDIR :転送
JP 8100H :転送したプログラムへジャンプ

ORG 0100H :8100hへ転送されて実行される
LD A, 008H :HC273 Q4
OUT (IOCE0), A :バンク切り替え

LD HL, 8200H :RAM1(アドレス 8200h)から
LD DE, 0200H :RAM2(アドレス 0200h)へ
LD BC, 7600H :7600hバイト
LDIR :転送
JP 0200H :転送したプログラムへジャンプ

ORG 0200H :リスタート RAM0
LD SP, STACK1

;
; 以下、プログラムが続く
;
END
```

リセット後はHC273のQ<sub>1</sub>~Q<sub>4</sub>がLレベルになるので、アドレス0000h~7FFFhにはROMが選択されます。Z80はリセット後、アドレス0000hから実行を開始するので、必ずROMが選択されなくてはなりません。よってHC273などのリセット機能を備えたラッチを使用します。

そしてプログラムAにより仮のスタック・エリアを設定し、BとCの領域をブロック転送命令でアドレス8000h以降のRAM領域に転送します。

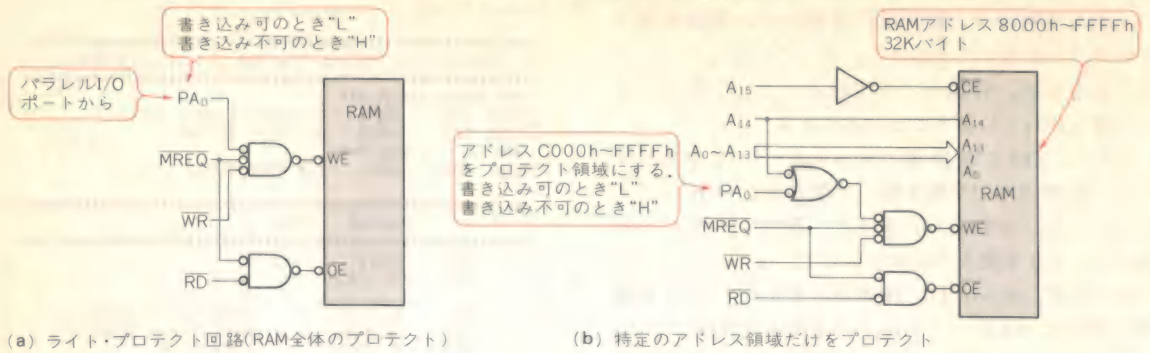
次にプログラムBでバンク・セレクト・ビットのQ<sub>4</sub>をHレベルにし、アドレス0000h~7FFFhの領域にRAM<sub>0</sub>を選択します。そして先ほどとは逆方向に、アドレス8000hからのデータをアドレス0000hの領域に転送します。これで64KバイトすべてRAMのシステムができます。

またこの回路では、アドレス8000h以降の領域もバンク切り替え可能です。RAM<sub>1</sub>~RAM<sub>4</sub>の切り替えは、HC273のQ<sub>1</sub>、Q<sub>2</sub>で選択します。

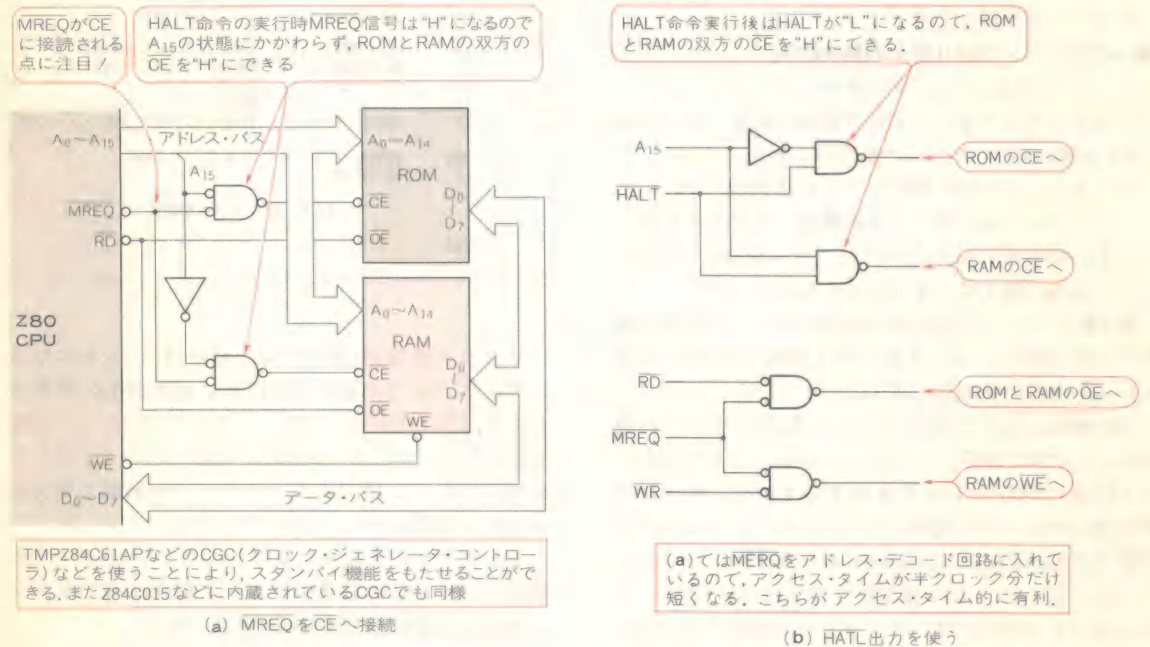
当然ながら、この領域にスタック・エリアを確保している場合は、アドレス0000h~7FFFhの領域に再確保しないと、バンク切り替えしたとたんにスタック領域がバンクの裏に消えてアクセスできなくなってしまう。

またこの図17の回路を具体的な回路図にしたのが図18で、この回路のコントロール・プログラム例を

〈図 19〉 ライト・プロテクト回路の例



〈図 20〉 スタンバイ回路



リスト 1 に示します。

### ● RAM のプロテクト回路例

通常は普通に読み書きができ、ある**特定の状態のときに書き込み禁止(プロテクト)**をかけられる回路の例を図 19 に示します。

これもバンク切り替え回路と同じように、任意の I/O 空間に平行・ポートをつけて、そのポートの任意のビットを書き込み可/不可フラグとして使用します。

この信号と  $\overline{\text{WR}}$  信号(実際には  $\overline{\text{MREQ}}$  と  $\overline{\text{WR}}$  の負論理 AND を取った  $\overline{\text{MWR}}$ )を負論理 AND をとって RAM の書き込み信号にすればよいわけです。

図 19 の例では、ポートのビットを H レベルにすると書き込み禁止となります。

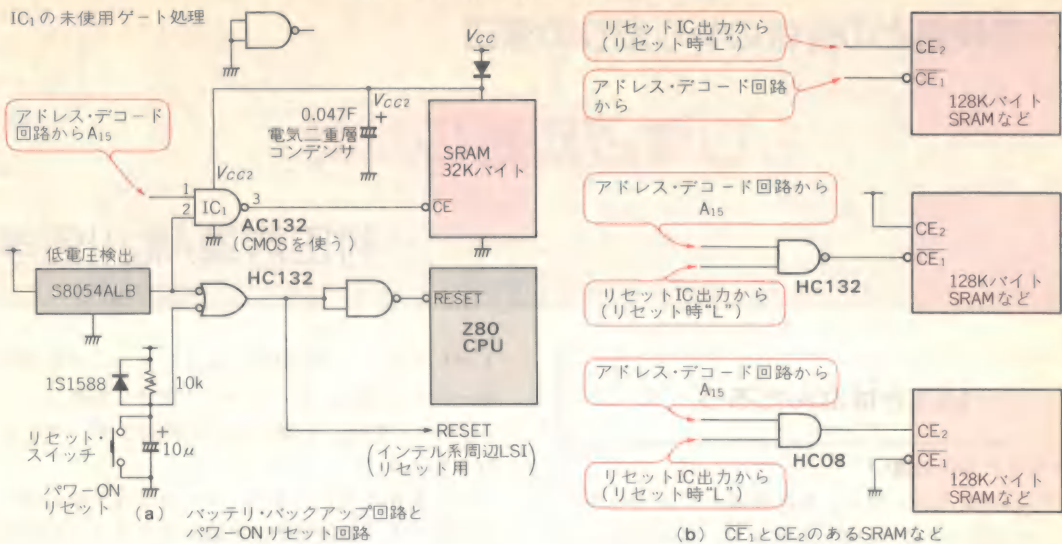
また場合によっては、任意のアドレス領域にだけプロテクトをかけたいときもあるでしょう。このときは図 19 (b)のように、プロテクトをかけたいアドレス領域をデコードしてプロテクト回路に入力すればよいわけです。

アドレスのデコード回路を変更すれば、任意のアドレス領域のプロテクトが実現できます。もっとも、任意といえども ROM 領域はもとと書き込みできないので意味はありませんが。

また、このプロテクト回路についてだけではありませんが、CPU からのコントロール信号とメモリとの間にゲートが入ると、それだけゲート遅延時間が増加することになります。その分だけ高速なアクセス・タイムのメモリを使うか、遅延時間の少ないゲートを使



〈図 21〉 RAM のバッテリー・バックアップ回路



うなどの工夫が必要です。

## 低消費電力モードとバックアップ

### ● バッテリーで動作させる

Z80 も CMOS 化されて省電力になり、マイコン・システム全体を**バッテリー動作**させることができるようになりました。専用のクロック・ジェネレータを使ってシステム全体に**スタンバイ機能**をもたせることにより、バッテリーの寿命を延ばすことができます。

低消費電力対応のメモリは、**CE が H レベルであれば非選択状態であると認識し、低消費電力モードに入ります。**

そこで図 20 のように、メモリのデコード回路に  $\overline{\text{MREQ}}$  信号を接続し、 $\overline{\text{RD}}$ 、 $\overline{\text{WR}}$  信号はメモリの  $\overline{\text{OE}}$  や  $\overline{\text{WE}}$  に直接接続します。スタンバイ時 (HALT 命令を実行したとき)、 $\overline{\text{MREQ}}$  は H レベルになるので、メモリをスタンバイ状態にすることができます。

CPU を HALT 命令で停止させた状態からもう一度プログラムを実行させるには割り込みを使いますが、ここでは説明しません。

ただしすでに説明したように  $\overline{\text{CE}}$  に  $\overline{\text{MREQ}}$  信号を入れるので、アクセス・タイムにはこのことを十分考慮に入れて設計してください。

図 20 (b) は、 $\overline{\text{MREQ}}$  ではなく **HALT** を使った例です。 $\overline{\text{MREQ}}$  は通常どおり  $\overline{\text{RD}}$  や  $\overline{\text{WR}}$  と接続しているので、アクセス時間的には **HALT** を  $\overline{\text{CE}}$  に接続するゲート分だけ気にすればよいので、(a) の回路よりはアクセス・タイムに余裕があります。

また ROM のアクセス・タイムを稼ぐために、 $\overline{\text{CE}}$  をグラウンドに接続しっぱなしにする方法は、当然の

ことながら、この低消費電力モードにはなりません。

### ● SRAM のバッテリー・バックアップ

継続的なデータ收拾システムや、RAM 領域に製作途中のプログラムをいれてデバッグする開発中のマイコン・システムでは、電源が瞬断した場合などに備えて SRAM の回路に、バッテリー・バックアップ回路を付加すると便利です。

またたいていの CPU は、電源の瞬断が起こると暴走し、RAM に誤書き込みしてしまいます。この誤書き込みも防止することができます。

このためには**電源検出用に便利な、専用のリセット IC**があります。

基本的には、バックアップ時に SRAM の  $\overline{\text{CE}}$  端子を H レベル (2 V 以上) になるようにし、別電源 (電池やコンデンサ) に切り替えます。

図 21 に、電圧低下の検出用に S805 シリーズを使った例を示します。パワー ON リセット回路と負論理の OR を取り、CPU にリセットをかけます。

この回路のポイントは、SRAM の  $\overline{\text{CE}}$  につな**がった AC132 の電源を電気二重層コンデンサから取り、SRAM と AC132 の両方をバックアップしている点**です。AC132 も動作させていることで、SRAM の  $\overline{\text{CE}}$  は H レベルに保たれるからです。

### ●参考文献●

- (1) 鈴木八十二：半導体 MOS メモリとその使い方，日刊工業新聞社。
- (2) 榎東芝，MOS メモリデータブック ROM 編，1991 年。
- (3) 榎東芝，MOS メモリデータブック RAM 編，1993 年。  
(トランジスタ技術 1994 年 6 月号に加筆，修正)

## 外部装置との情報のやり取りの窓口

### I/O周辺回路の設計

村田浩義/北川信孝

#### I/O とはなんだろう

##### ● メモリとI/Oの違い

マイコン・システムなら、外部装置とデータのやり取りをする窓口が必ず必要になります。この**外部とデータをやり取りする窓口にあたる部分をI/O**と呼びます(図1)。メモリもデータ・バスに直結されますが、外部装置とデータのやりとりはしないのでI/Oとはいいません。

第1章でも説明したように、CPUに接続するメモリとI/Oの接続回路の違いは、おおざっぱにいうと、基本的にはMREQがアクティブになるかIORQがアクティブになるかの違いだけです。アドレス・バスが出力され、RDまたはWRにより読み出しか書き込みの選択をする部分は、メモリについてもI/Oについても基本的に同じです。

しかしソフトウェア的な立場から見たときには、非常に大きな違いがあります。

##### ● I/O デバイスとは

マイコン・システムでは、各種バスが重要な働きを

しています。アドレスをセットしデコード回路で相手を一だけ選び、コントロール信号で制御しながらデータ・バスを通じて選択された装置とデータのやりとりをします。

バスを使うことにより、マイコンは大量のデータの受け渡しを、装置間で信号線を増やすことなく容易に行えます。

この反面、バスは時分割でしか使えませんが、**外部装置側にデータをラッチする機能とバスをバッファリングする機能**が必要になります(図2)。

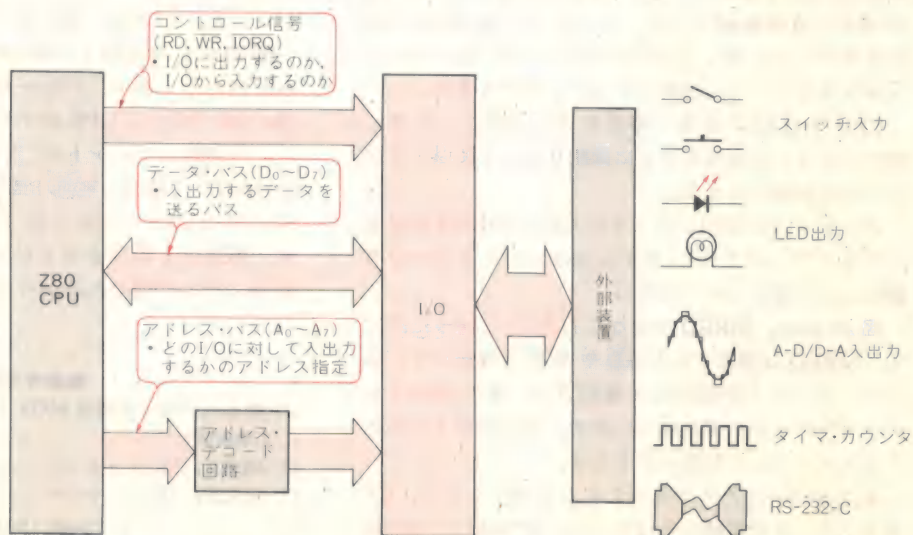
このように、CPUが外部のデータを読み込みたいときに、外部からのデータをデータ・バスに出力するもの、またCPUが外部にデータを出力したいときに、データ・バスからデータを入力して外部に出力するものなど、**入出力をコントロールするラッチやバッファがI/Oデバイス**なのです。

#### Z80のI/Oアクセス・タイミング

##### ● I/O リード・サイクル

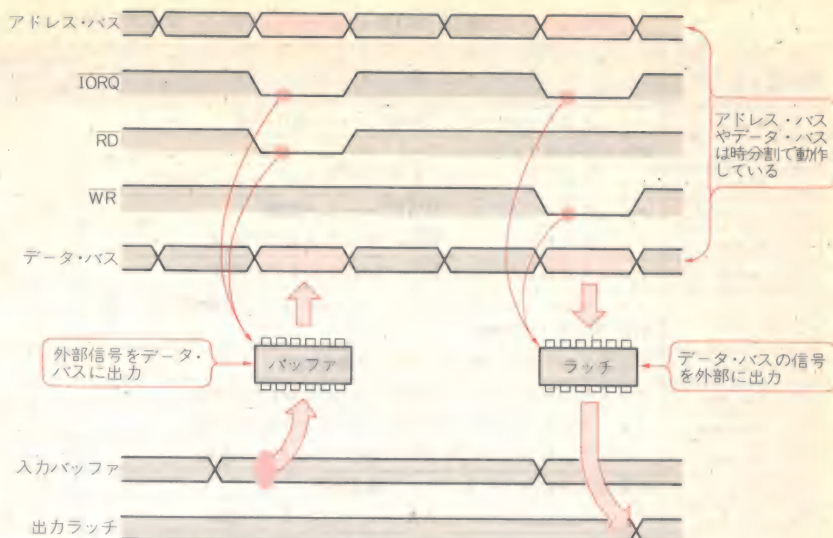
I/Oに対するアクセスは、**I/O制御命令であるIN/**

〈図1〉  
I/Oとは

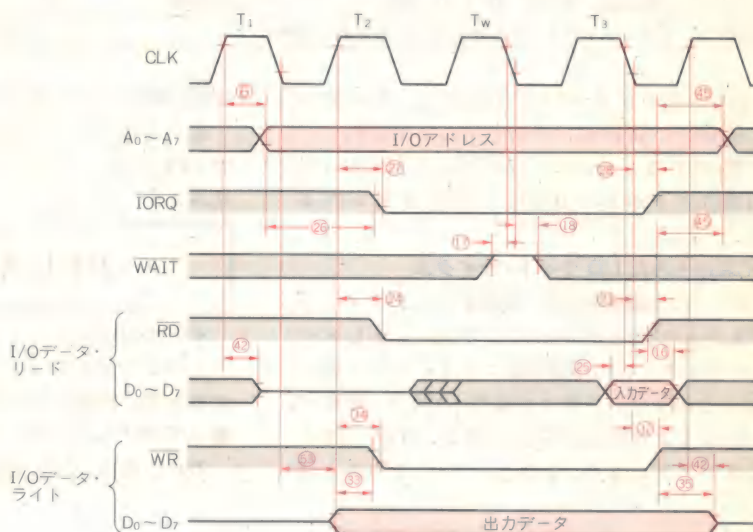




〈図2〉  
I/O デバイスとバスの関係



〈図3〉  
Z80のI/O リード・ライト・  
サイクルのタイミング



OUT/OTIR/INIRなどの命令が実行されたときに発生します。

Z80のI/Oリード・ライト・サイクルを図3に、アクセス・タイミングを表1に示します。Z80のI/Oリード/ライト・サイクルは、図中のTwで示すウェイト・サイクルが、CPU自身によって強制的に1クロック挿入されるので、4クロック・サイクルで実行されます。

IN命令の実行、つまりI/Oリード・サイクルでは、まずT<sub>1</sub>の立ち上がりでI/Oポート・アドレス(A<sub>0</sub>~A<sub>7</sub>)がアドレス・バスに出力され、T<sub>2</sub>の立ち上がりでIORQとRDがアクティブになります。

そしてTwによる1クロック分のウェイトが入ったあと、T<sub>3</sub>の立ち下がりCPUはI/Oポートから出力

されたデータを取り込みます。

#### ● デバイスからみたI/Oリード・サイクル

メモリ・リード・サイクルではT<sub>1</sub>の立ち下がり出力されていたMREQやRDが、I/Oリード・サイクルでは半クロック遅くなり、T<sub>2</sub>の立ち上がりで出力されます。しかしTwによるウェイトが1クロック入るので、実質的にはT<sub>2</sub>とTwそしてT<sub>3</sub>の立ち下がりまでと、アドレスが確定してから3.5クロック、IORQがアクティブになってから2.5クロック分の時間があり、メモリ・リード/ライト・サイクルよりアクセス時間に余裕があります。

#### ● I/Oライト・サイクル

OUT命令の実行、つまりI/Oライト・サイクルでは、I/Oリード・サイクルと同様にT<sub>1</sub>の立ち上がりで

〈表1〉 Z80 の I/O リード・ライト・サイクル(単位 ns)

番号	記号	項目	6MHz 版		8MHz 版		10MHz 版		20MHz 版	
			最小	最大	最小	最大	最小	最大	最小	最大
6	$t_{dCP(A)}$	クロック立ち上がりからの有効アドレス出力遅延		90		80		65		57
16	$t_{hD(RDY)}$	RD 立ち上がりに対するデータ・ホールド時間	0		0		0		0	
17	$t_{sWAIT(CF)}$	クロック立ち下がりに対する WAIT 信号セットアップ時間	60		50		20		7.5	
18	$t_{sWAIT(CF)}$	クロック立ち下がり後の WAIT ホールド時間	10		10		10		10	
23	$t_{dCF(RDY)}$	クロック立ち下がりから $\overline{RD} = "H"$ になるまでの遅延		70		60		55		40
24	$t_{dCF(CF)}$	クロック立ち上がりから $\overline{RD} = "L"$ になるまでの遅延		70		60		55		40
25	$t_{sD(CF)}$	クロック立ち下がりに対するデータ・セットアップ時間(M2, M3, M4, M5 サイクル時)	40		30		25		12	
26	$t_{dA(IORQ)}$	$\overline{IORQ}$ 立ち下がりから先立つアドレス確定時間	107		75		50		0	
27	$t_{dCF(IORQ)}$	クロック立ち上がりから $\overline{IORQ} = "L"$ になるまでの遅延		65		55		50		40
28	$t_{dCF(IORQ)}$	クロック立ち下がりから $\overline{IORQ} = "H"$ になるまでの遅延		70		60		55		40
32	$t_{dCF(WR)}$	クロック立ち下がりから $\overline{WR} = "H"$ になるまでの遅延		70		60		55		40
33	$t_{dD(WR)}$	$\overline{WR}$ 立ち下がりから先立つデータ確定時間	-55		-55		-10		-30	
34	$t_{dCF(WR)}$	クロック立ち上がりから $\overline{WR} = "L"$ になるまでの遅延		60		60		50		40
35	$t_{dWR(D)}$	$\overline{WR} = "H"$ になってからの出力データ保持時間	30		15		10		0	
42	$t_{dCF(DZ)}$	クロック立ち上がりからデータ・バス・フロート状態までの遅延		80		70		65		40
45	$t_{dCF(A)}$	MREQ, $\overline{IORQ}$ , RD または $\overline{WR}$ からのアドレス保持時間	35		20		20		0	
53	$t_{dCF(D)}$	クロック立ち下がりからデータ出力までの遅延		130		115		110		75

I/O アドレスがアドレス・バスに出力され、 $T_2$  の立ち上がりで  $\overline{IORQ}$  と  $\overline{WR}$  がアクティブになります。

そして  $T_w$  による 1 クロック分のウェイトが入ったあと、 $T_3$  の立ち下がりでも  $\overline{IORQ}$  と  $\overline{WR}$  を H レベルにします。

#### ● デバイスからみた I/O ライト・サイクル

アドレス・バスの確定時間も  $\overline{IORQ}$  の出力タイミングも、I/O リード・サイクルと同じです。しかしメモリ・ライト・サイクルでは  $\overline{MREQ}$  より 1 クロック遅れて  $T_2$  の立ち上がりでアクティブになっていた  $\overline{WR}$  が、I/O ライト・サイクルでは  $\overline{IORQ}$  と同じく  $T_2$  の立ち上がりでアクティブになり、I/O リード・サイクルとほ

ぼ同じ動作となります。

また I/O デバイスは、 $\overline{WR}$  クロックの立ち上がりで CPU からデータ・バスに出力されたデータを取り込みます。

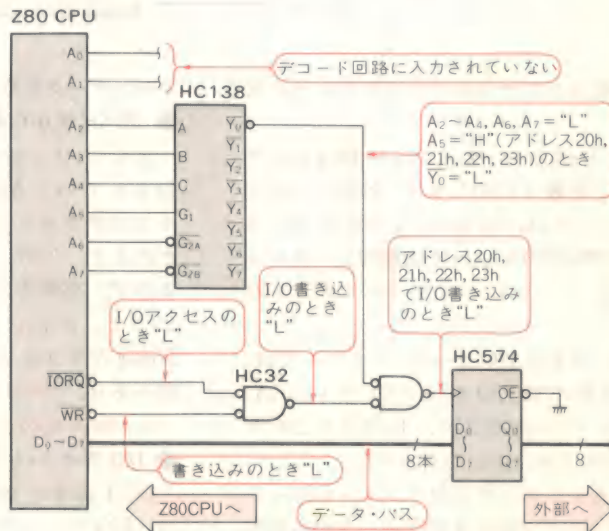
### アドレス・デコードとは

すでに説明したように、I/O デバイスの基本はラッチとバッファです。ここではラッチとバッファで構成する非常に簡単な I/O について説明します。

#### ● I/O アドレス・デコード回路

Z80 の場合、I/O 空間はアドレス・バス  $A_0 \sim A_7$  で指

〈図4〉  
8 ビット・パラレル出力ポートの回路の例





定できる 0~255 までの 256 個です。Z80 では本来未定義な 16 ビット I/O アドレスを使うなら  $A_0 \sim A_{15}$  を使いますが、ここでは基本である 8 ビット I/O アドレスで説明します。

256 個のアドレスの中から、選択したいアドレスに該当する I/O デバイスだけをアクティブにするのが I/O アドレス・デコーダです。

I/O アドレス・デコードに HC138 を使い、出力ラッチに HC574 を使った I/O ポートの回路を図 4 に示します。

すでに説明したように、Z80 の I/O 空間の選択は  $\overline{IORQ}$  を L レベルにすることで制御しています。このため I/O デバイスの選択には  $\overline{IORQ}$  信号を負論理 AND する必要があります。

この回路では  $\overline{IORQ}$  と  $\overline{WR}$  を負論理 AND した  $\overline{IOWR}$  (I/O 書き込み要求) をつくり、これと HC138 でデコードしたセレクト信号を AND して HC574 の書き込みクロックにしています。

#### ● HC574 のアドレス

HC138 には A~C 入力と、 $G_1$ 、 $\overline{G_{2A}}$ 、 $\overline{G_{2B}}$  の三つのイネーブル端子があります。つまり図 4 では、 $A_7$  と  $A_6$  が L レベル、 $A_5$  が H レベルのときに、 $A_2 \sim A_4$  の状態で  $\overline{Y_0} \sim \overline{Y_7}$  が選択されます。また  $A_0$  と  $A_1$  はデコード回路に入力していません。

以上より HC574 は、20h、21h、22h、23h のどのアドレスでも選択することができます。また HC138 のイネーブル端子に  $\overline{IORQ}$  を接続しているわけではないので、HC138 の  $\overline{Y_0}$  端子は I/O 命令以外のメモリ・アクセスやリフレッシュのときにもアクティブになることがある点に注意します。

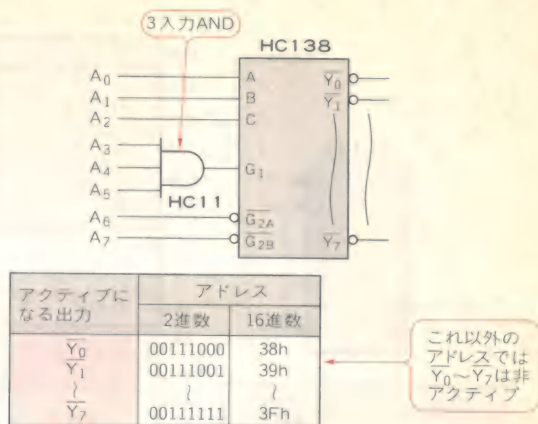
これでは I/O デバイスであるはずの HC574 が、I/O 命令以外のときでも選択されてしまいそうです。しかし  $\overline{IOWR}$  は  $\overline{IORQ}$  と  $\overline{WR}$  がアクティブになるとき、つまり OUT 命令を実行したときにアクティブになります。この信号と AND を取っているので、20h、21h、22h、23h の I/O アドレスに対して OUT 命令を実行するときだけ書き込みクロックが出力されるのです。

#### ● アドレスのフル・デコードとは

一つの I/O デバイスに対して複数のアドレスからアクセスできるのは、たしかに気持ち悪いかもしれませんが、複数のアドレスで同じ I/O が選択されてしまうのは、 $A_0$  と  $A_1$  をデコード回路に入力していないことに原因があります。

つまり  $A_0$  と  $A_1$  が “0” でも “1” でも HC138 の  $\overline{Y_0}$  が L レベルになるからです。アドレスをビットで表現すると 001000XXb となります。この XX のことを Don't Care といいます。またこのようなアドレスをイメージ・アドレスともよびます。

〈図 5〉 8 ビット I/O アドレスをフル・デコードした例



どうしてもただ一つのアドレスで選択できなければだめだという場合は、図 5 のように  $A_0$  と  $A_1$  もデコード回路に接続します。これをフル・デコードといいます。

しかしつないでいる I/O デバイスが数個で、あとから別の I/O を拡張することもないのであれば、わざわざゲートを使ってフル・デコードする必要はありません。また部品の節約にもなります。

フル・デコードは例えば 100 個以上もの I/O デバイスを接続するときに必要になります。Z80 の I/O 空間は基本的に 256 バイトですから、一つの I/O デバイスに四つのアドレスが振られていたのでは、アドレスが足りなくなってしまいます。

フル・デコードされていない I/O の場合は、一般的には Don't Care のビットに 0 を入れたアドレスでアクセスします。

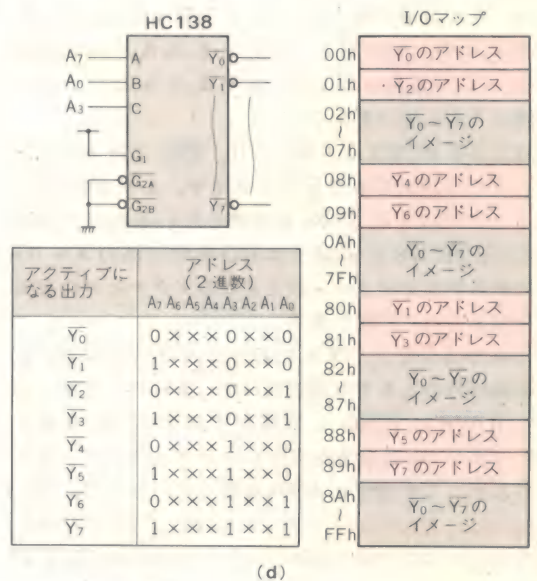
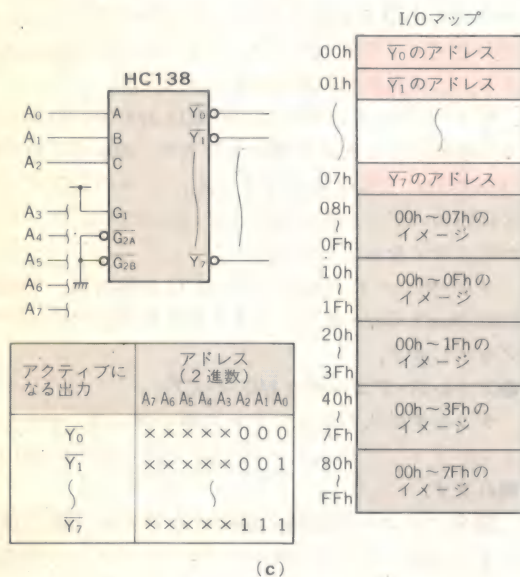
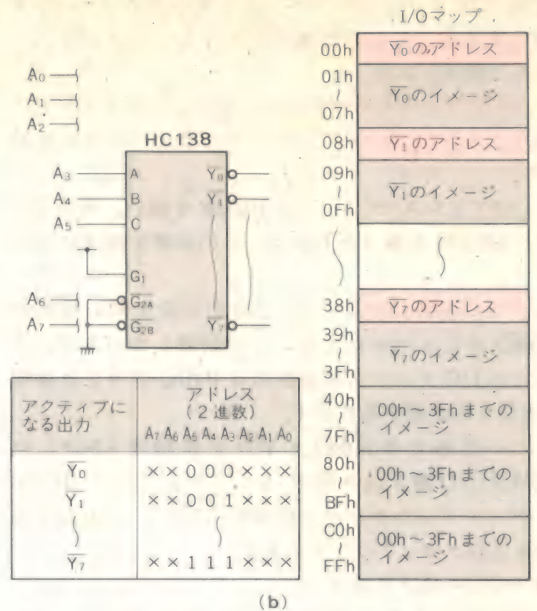
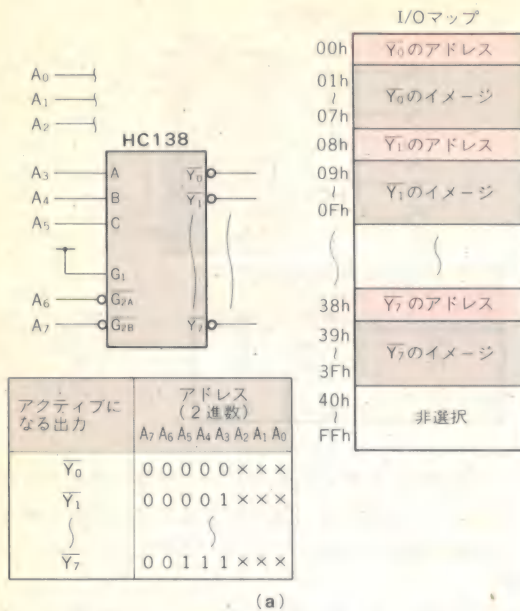
#### ● アドレス・デコードと I/O マップ

デコードに入力するアドレス線の違いは、I/O デバイスのアドレスやイメージ・アドレスの発生の違いに現れます。

図 6 にデコード回路に接続するアドレス線の違いによる I/O アドレスとそのイメージ・アドレスの違いを示します。  $\overline{Y_0}$  を I/O アドレス 00h でアクティブにしたいなら、図 6 (a) のように、 $G_1$  を H レベルにして  $A_3 \sim A_7$  でデコードします。この場合、01h~07h まで 00h のイメージ・アドレスが発生します。  $\overline{Y_1}$  がアクティブになるのは 08h で、09h~0Fh までがそのイメージになります。そしてアドレス 40h~0FFh までは  $\overline{Y_0} \sim \overline{Y_7}$  のどれもアクティブにはなりません。

では、 $\overline{G_{2A}}$  と  $\overline{G_{2B}}$  に接続している  $A_7$  と  $A_6$  を外して入力をグラウンドに落とすとどうなるでしょう [図 6 (b)]。  $A_7$  と  $A_6$  を外したのでこのビットも Don't Care になります。しかし  $\overline{Y_0} \sim \overline{Y_7}$  のアドレスは変化していません。

〈図6〉 アドレス・デコードとI/O マップの例



しかし図(a)では 40h 以降のアドレスではすべて非選択になったものが、**ふたたび $\overline{Y_0}$ がアクティブ**になります。

このように下位ビットが Don't Care の場合は、同じ I/O が続けて選択されるようにイメージが発生しますが、上位ビットが Don't Care の場合は、それ以下のアドレスを繰り返すようにイメージが発生します。

また、いままでの説明では HC138 の A~C 入力端子にはアドレス・バスを A<sub>2</sub>~A<sub>4</sub> のように順番に接続していました。もしこの順番が逆、または A<sub>7</sub> を A 入力

に、A<sub>0</sub> を B 入力に入れるなどしたらどうなるでしょうか。図 6 (d) にその例を示しますが、 **$\overline{Y_0} \sim \overline{Y_7}$  のセレクト線がアドレスに対して順番に選択されず**、ばらばらになってしまいます。

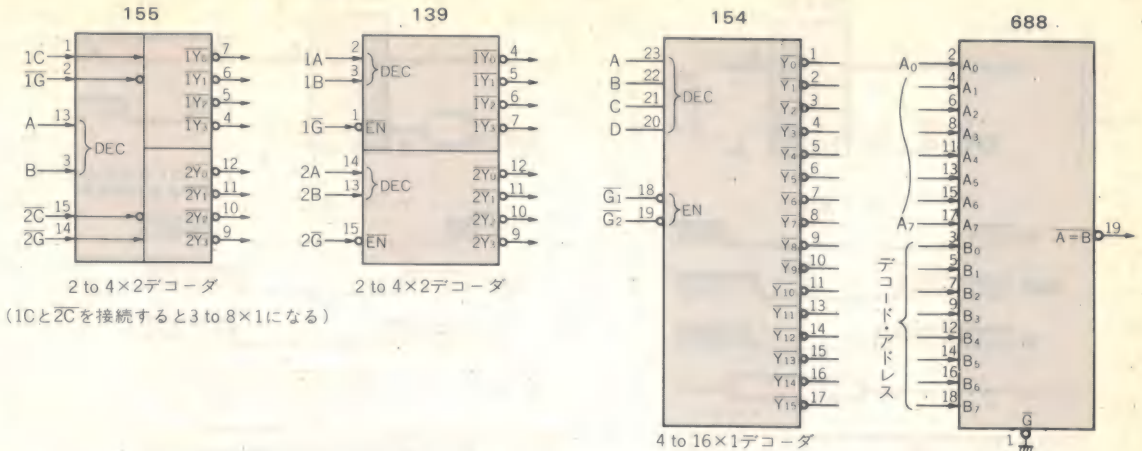
もちろんこうしてもアドレスの指定を間違わなければ動作しますが、特に理由のない限り、順番に接続したほうがアドレスのセレクトも順番になりわかりやすくなります。

#### ● HC138 以外のデコード IC

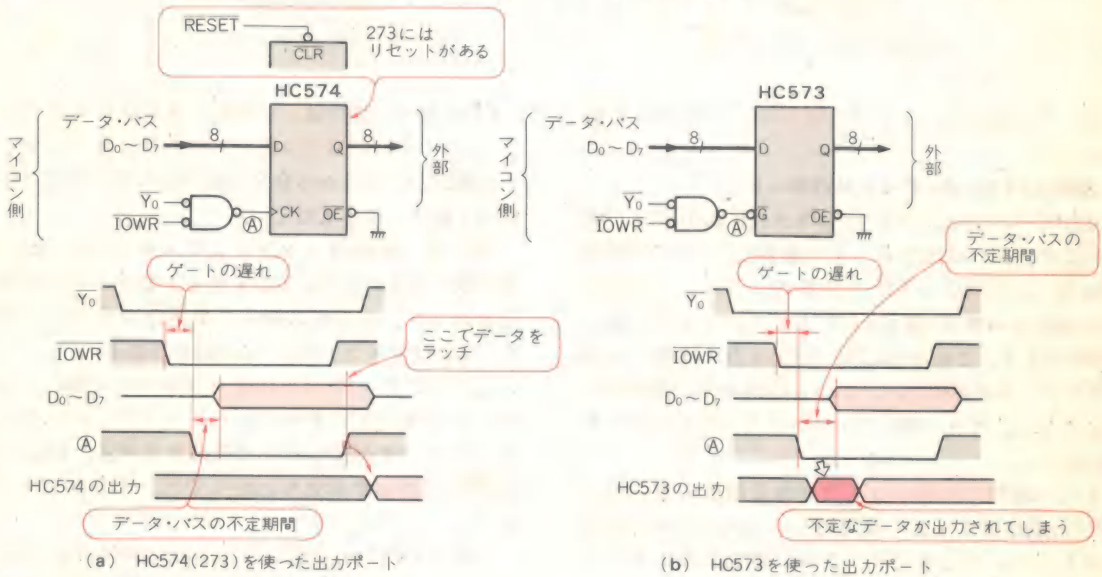
アドレス・デコードに使える IC は、HC138 以外に



〈図7〉よく使われるデコード IC



〈図8〉 パラレル・データ出力ポートの例



も図7に示すようなものがあります。

HC138の出力は8本なので、8個のI/Oデバイスを接続することができます。8個以上のデバイスを接続したいときは、HC154を使ったりデコードICを組み合わせたりします。

またHC688(8ビット・コンパレータ)を使うと、B入力側の状態を変えるだけで任意のアドレスを選択することもできます。

## 基本的なデータ入出力ポートの設計

### ● 簡単なパラレル・データ出力ポート

いちばん基本的な8ビットのパラレル出力ポートとしては、すでに図4に回路例を示しました。このよ

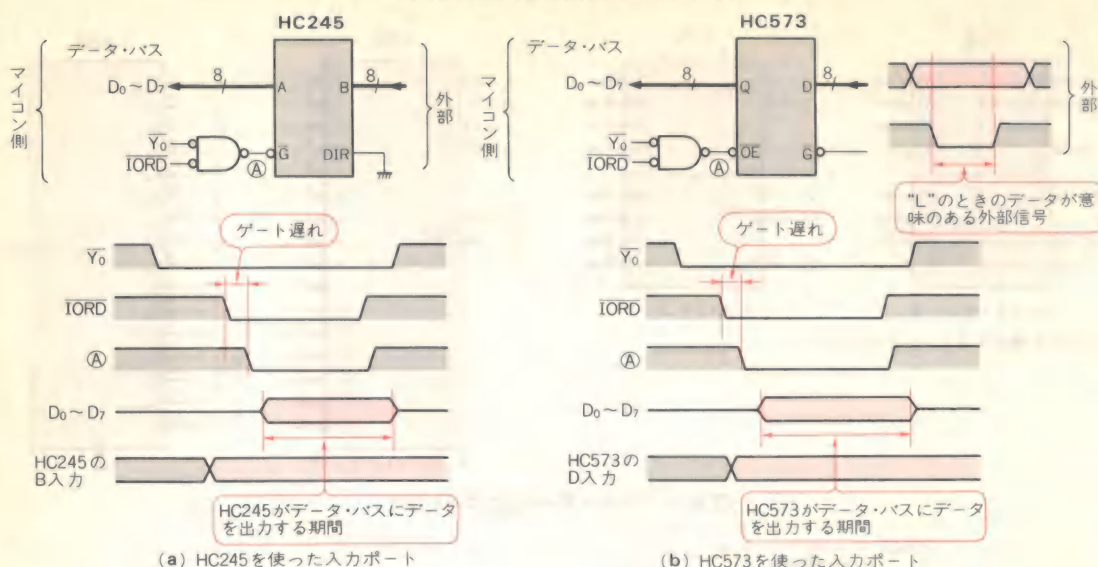
うな出力ポートには8ビットのDラッチ(HC574、HC374)がよく使われます。

ここで注意したいのは、トランスペアレント型ラッチであるHC573やHC373は不向きである点です。表1のZ80のI/Oリード/ライト・サイクルをみるとわかるように、WRの立ち下がりでデータ・バスのデータがまだ確定していません。

HC573などではWRが立ち下がった時点でデータ・バスのデータを取り込んで外部に出力してしまうので、図8(b)に示すように不定データが出力されてしまうことになります。

またHC574はリセット端子がなく、電源投入直後は不定データが出力されます。ものによってはリセット直後出力がクリアされていないとまずい場合があります。

〈図 9〉 パラレル・データ入力ポートの例



(a) HC245を使った入力ポート

(b) HC573を使った入力ポート

ます。その場合は、リセット端子付きの HC273 を使います。

### ● 簡単なパラレル・データ入力ポート

入力ポートでは、バス・バッファ (HC245) がよく使われます。これは出力ポートで接続していた  $\overline{IOWR}$  ではなく、 $\overline{IORD}$  と  $\overline{RD}$  を負論理 AND した  $\overline{IORD}$  (I/O 読み込み要求) 信号を、アドレスのセレクト線と AND します。これを HC245 の  $\overline{G}$  入力に接続すれば、I/O アドレスが選択され、かつ I/O 読み出し要求が出たときだけ、ゲート用のパルスが出力されるわけです (図 9)。

また一般的に外部から入力される信号の変化と、CPU の動作はまったく非同期です。例えばある信号線が L レベルのときにデータ線に意味のあるデータが出力される装置からデータを読み込みたいときは、HC573 を使いそのデータをラッチします。こうすることで、その装置が常時データを出力しなくても、有効なデータを保持しておくことができ、CPU はいつでもそのデータを読み込むことができます [図 9 (b)]。

### ● OUT 命令実行時のバスの動作

OUT 命令を実行したときの具体的なバスの動きを図 10 に示します。ここでは HC273 の  $Q_1$  ピン (ビット 0) にモータを接続してあるとします。モータはビット 0 が “1” になると回転します。また電源投入後すぐにモータが回転するとたいへんなので、HC273 のクリア端子にリセット信号を接続しています。

プログラムは、ビット 0 でモータの制御を行うので、A レジスタに 01h (モータの ON データ) を格納し、モータ・コントロールの I/O アドレス (回路ではアドレス 00h) に OUT 命令を実行します。

OUT 命令の実行は、いきなり  $\overline{IORD}$  による I/O ライト・サイクルが始まるわけではありません。マイコンの動作はメモリから命令を読み込んで、それにしたがって動作しているのです。

つまり、 $\overline{MREQ}$  と  $\overline{RD}$  によりメモリから OUT 命令が読み込まれ、次に OUT する I/O アドレスが読み込まれます。この次に初めて  $\overline{IORD}$  がアクティブになり、I/O ライト・サイクルが始まります。

こうしてデータ・バスには A レジスタの値、つまりモータ ON のデータが出力されます。しかしこれですぐにモータが回るわけではありません。HC273 は  $\overline{IOWR}$  クロックの立ち上がりでデータをラッチします。

つまり CPU が  $\overline{WR}$  を立ち上げた瞬間に HC273 はデータをラッチし、これが初めて外部に出力されモータが回りだすのです。

### ● IN 命令実行時のバスの動作

IN 命令を実行したときの具体的なバスの動きを図 11 に示します。ここでは HC245 の B 入力にスイッチを接続してあるとします。

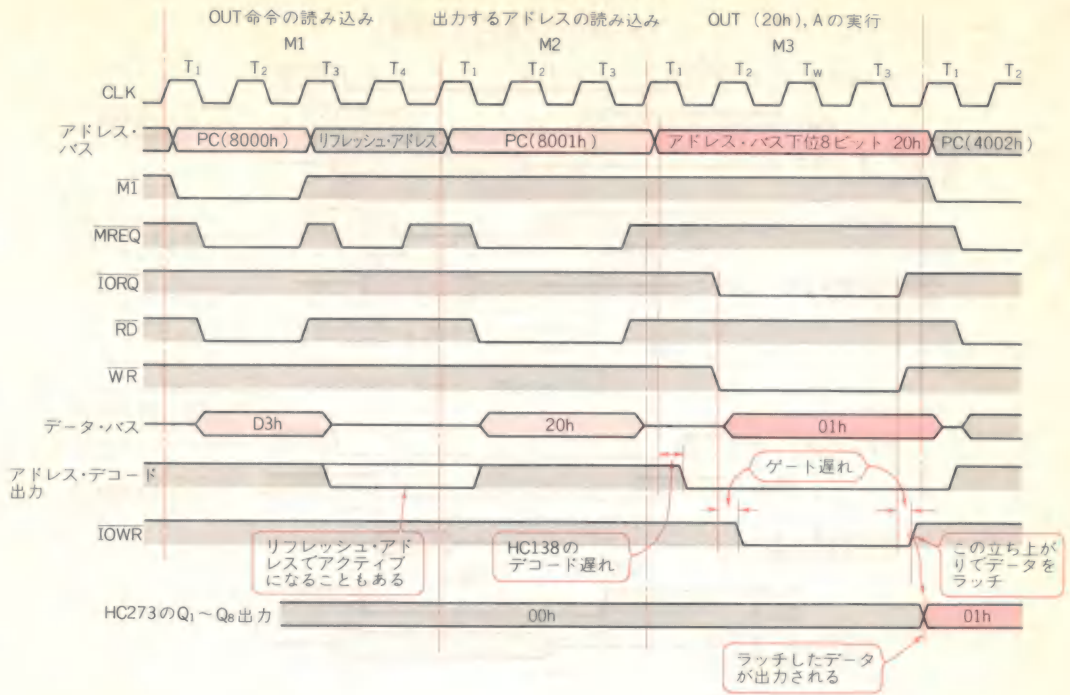
これも OUT 命令同様、いきなり I/O リード・サイクルが始まるのではなく、IN 命令とデータを入力する I/O アドレスをメモリから読み込んだあとに始まります。

I/O リード・サイクルが始まると、アドレス・バスには指定された I/O アドレスが出力され、 $\overline{Y_1}$  がアクティブになります。そして  $\overline{IORD}$  が L レベルになるので、HC245 の  $\overline{G}$  入力がアクティブになり、データ・バスにスイッチの状態が出力されます。

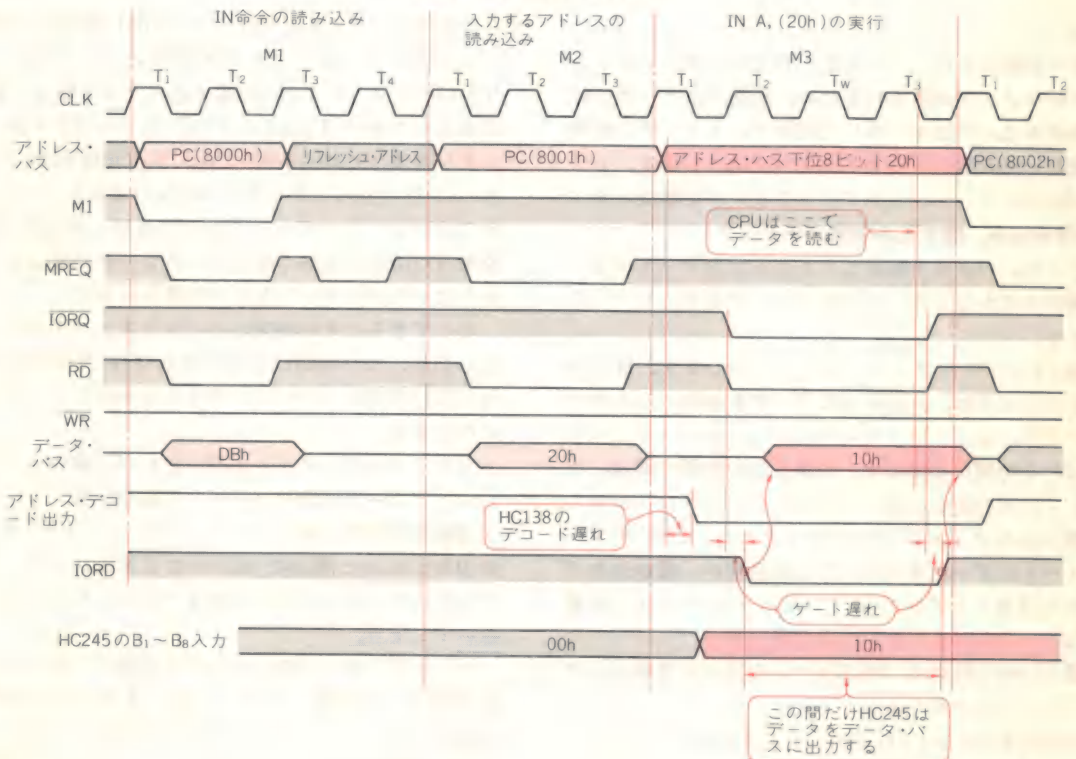
CPU はそのデータを読み込み、A レジスタに格納



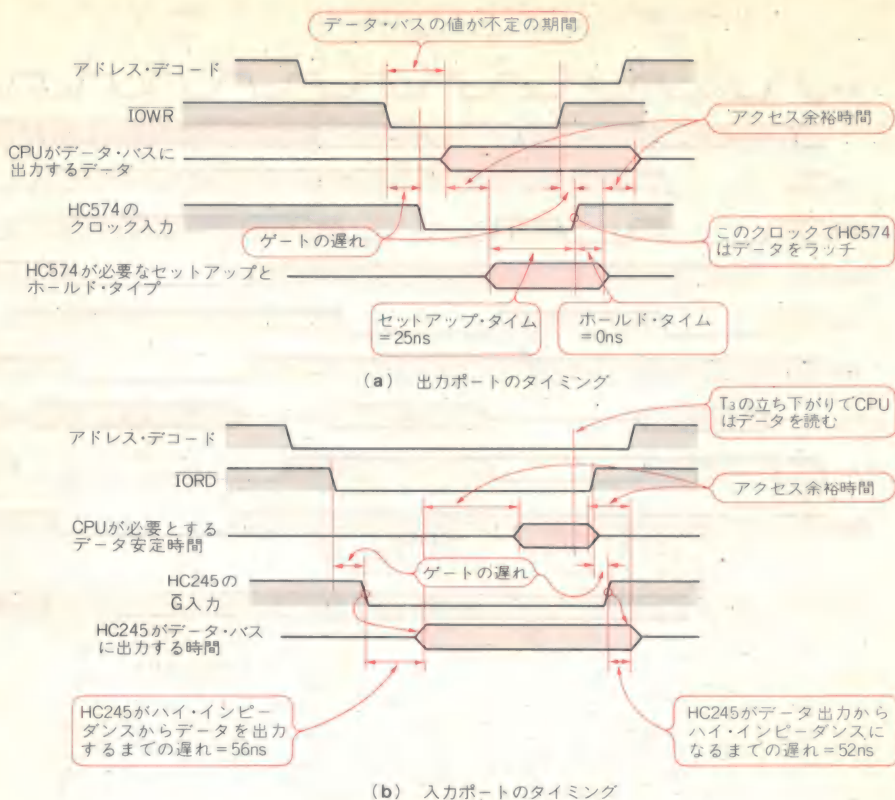
〈図 10〉 OUT 命令のバスの動作



〈図 11〉 IN 命令のバスの動作



〈図 12〉 アクセス・タイミングのチェック



します。

また余談ですが、スイッチ入力では必ずチャタリングというノイズが含まれるため、何度かスイッチの状態を読み込んで値が一致した段階で、スイッチの状態が確定したと判断するようにプログラムします。もしくは完全にハードウェアでチャタリングを除去します。

### ● 読み込み、書き込み時間のチェック

ここで、いままで設計してきた入出力ポートが正しく動作するかどうか、そのタイミングをチェックしてみます。

図 12 にアクセス・タイミングを示します。HC574 などのラッチは、入力されたデータを確実にラッチするために、**セットアップ・タイム**と**ホールド・タイム**と呼ばれる時間が必要です。つまりこの時間の間は、**データ・バスの値を一定**しておかなければなりません。

書き込みタイミングのチェックとは、CPU がデータ・バスにデータを出力している時間が、書き込むデバイスのセットアップ・タイムとホールド・タイムを満足しているかどうかを調べる作業です。

図 12 (a) の例では、HC574 に入力される書き込みクロックの立ち上がりより**セットアップ・タイム分だけ前の時間までに、CPU がデータ・バスにデータを出力しているかどうか**がポイントになります。

ここではシステム・クロック 10 MHz 動作の Z80 について考えてみます。まず  $\overline{\text{IOWR}}$  のパルス幅、つまり I/O ライト・サイクルの書き込みパルス幅は、 $T_2$  の立ち上がりから  $T_3$  の立ち下がりまでの 2.5 クロック分あります。しかし実際に  $\overline{\text{WR}}$  や  $\overline{\text{IORQ}}$  が出力されるのは  $T_2$  の立ち上がりから 50 ns の遅延があります。さらにアドレス・デコーダからの信号と負論理 AND を取った信号が実際の書き込みクロックになるので、さらにその分のゲート遅れが加算されます。

そしてさらにその時間からセットアップ・タイムを差し引き、まだ時間に余裕があれば(計算結果が負にならなければ)、アクセス・タイムを満足している結果になります。

もしアクセス・タイムが間に合わない場合は、アドレス・デコーダ回路などのゲート数を少なくし、ゲート通過遅延時間を減らしたり、セットアップ・タイムの少ないもっと高速な AC574 などを使います。それでもだめなときは CPU の動作クロックを下げるしかないでしょう。

ホールド・タイム側についても同様に、各ゲート遅延時間などを考慮してチェックします。ただ HC574 の場合はホールド・タイムが 0 ns ですから、ほとんどの場合問題ないでしょう。



次に図 12 (b)の読み込みアクセス・タイムについて考えます。これもさきほどのラッチと同様で、CPU がデータを読み込むにはセットアップ・タイムとホールド・タイムの時間だけ、データ・バスの値が一定でなければなりません。

10 MHz の Z80 の I/O リード・サイクルでは、RD が  $T_2$  の立ち上がりから 55 ns 遅れます。また HC245 の読み出しクロックは、CPU の  $\overline{\text{IORQ}}$  と RD、そしてアドレス・デコーダの出力を負論理 AND した信号になるので、これらのゲート遅延時間も加算されます。

そして HC245 に読み出しクロックが入力されてから、実際にデータ・バスにデータが出力されるまでの遅延時間も考慮に入れる必要があります。以上を計算しても、CPU が必要とするセットアップ・タイム以上の時間があればだいじょうぶです。

### 入力専用と出力専用のポート

#### ● 出力専用ポートで読み込みをしたら

出力に HC574 を入力に HC245 を使ったままでの例では、それぞれ別々のアドレスに対して出力ポ

ートと入力ポートを割り振っていました。

もしここで、HC273 の I/O アドレスに対して入力動作をしたらどうなるでしょう。 $\overline{\text{Y}_0}$  はアクティブになりますが、 $\overline{\text{IOWR}}$  がアクティブにならないので、HC273 に書き込み動作は行われません。また当然データを出力するデバイスも存在しないわけですから、CPU は誰もデータを出力していないデータ・バスから無意味なデータを読み込みます。

逆に HC245 の I/O アドレスに対して出力動作をしても、 $\overline{\text{Y}_1}$  はアクティブになりますが、 $\overline{\text{IORD}}$  がアクティブにならないので、HC245 はデータ・バスにデータを出力しません。また CPU が出力したデータは誰も受け取ることなく消えてしまいます。

#### ● 出力専用ポートではデータを保存する

このように、HC273 だけを接続した I/O アドレスは出力専用で、その I/O アドレスに出力したデータを再び読み込むことはできません。このような出力専用ポートにデータを出力するときは、出力したデータを後で調べられるように、RAM にワーク・エリアを設定して出力したデータを保存しておくという方法を取ります。

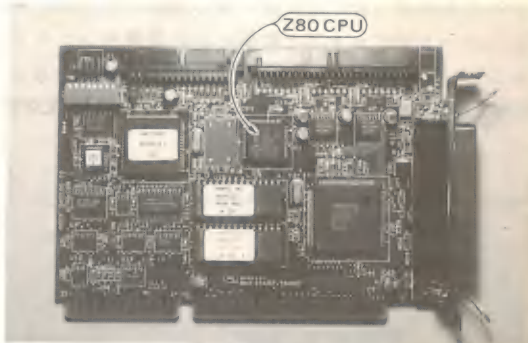
## Z80 はまだまだ現役

DOS/V ユーザでもある筆者は、ようやく昨年の年末に SCSI ボードで標準となったアダプティク社の AHA1542CF と CD-ROM ドライブを購入しました。ふとボードを見ると、なんと Z80CPU が搭載されているのです(写真 A)。

Z80 も CMOS 化され高速になり、日本では ASSP である Z84C011 や Z84C015 がよく売れているそうです。Z80 もなんだかんだ言われながら、たいていのシステムは実現できる能力をもっています。メモリの問題も少々めんどろですが、バンク切り替えで何とかできます。高速処理が必要な部分には DSP を速い CPU の代わりに使おうとしているくらいで、ほとんどのことは筆者もいまだに Z80 です。

Z80 を使う魅力の一つに開発ソフトの安さがあります。30 半ばになった筆者も、最近でははんだごてを持つよりソフトを作るほうが楽になりました。ソフトも FORTRAN, BASIC, アセンブラ (Z80, 8085), C と移ってきました。しかし C++ にはなかなか移行できませんが…。

PC8001 の拡張ボックスに PPI だけでボードを作り、漢字 ROM を接続して BASIC の IN と OUT 命令だけで動作させたときの感動が、マイコンとの付き合いの始まりでした。BASIC の動作速度の遅



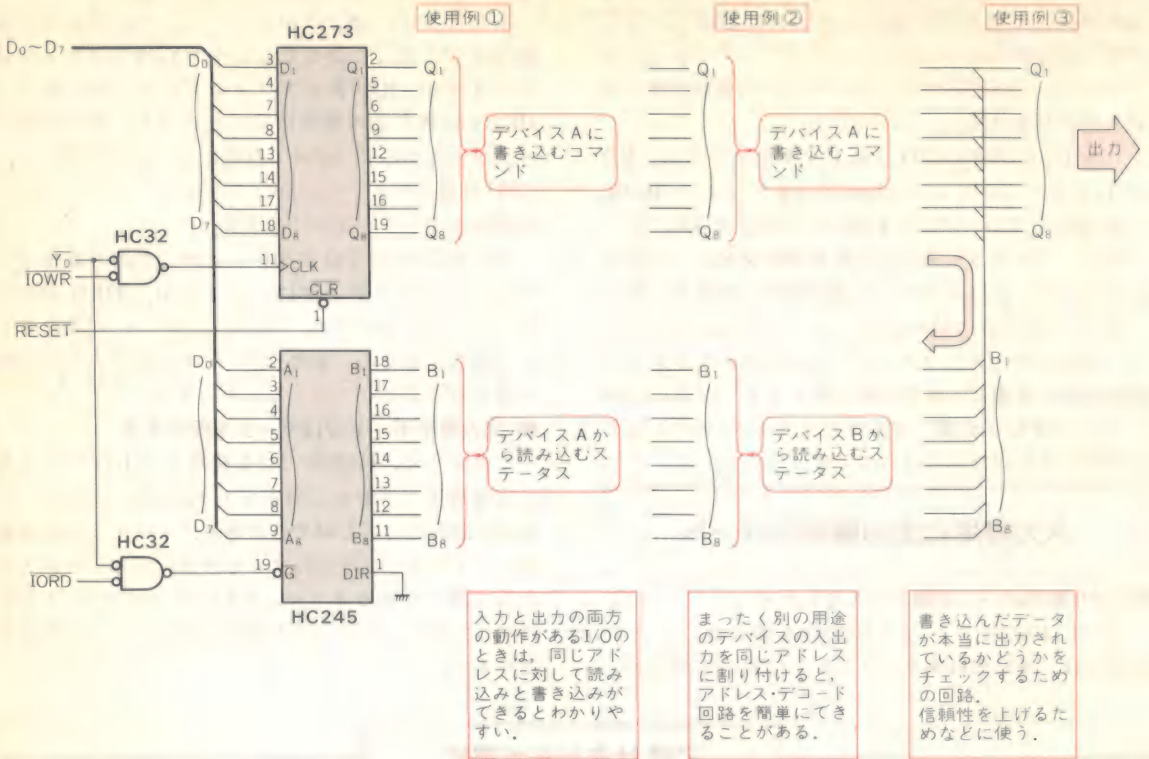
〈写真 A〉 Z80 が使われている SCSI ボード

さがアセンブラを身につけさせ、PAL ライタがマイコン・ボードを身につけさせてくれました。

言語が C になっただけで、DOS/V マシンでも PC8001 と同様のことができると知り、ドータ・ボードを買ってやってみるところです。EDLIN と PPI の達人を自称する筆者も、まだまだ Z80 と同様で現役のようです。

〈村田浩義〉

〈図 13〉 同じ I/O アドレスを使った入力、出力ポート例



データを保存しなければならない理由としては、その出力ポートがビットごとに何らかの装置をコントロールしているときなどのためです。例えばビット 0 に LED の ON/OFF を、ビット 7 にモータの ON/OFF

#### 〈リスト 1〉 出力専用ポートでのデータ保存法

```

CTRL_A: EQU    80H    ;コントロール・ポート
                ;bit0 "1" LED ON
                ;bit7 "1" MOTER ON

;      初期化ルーチンの途中
LD      A, 0
OUT     (CTRL_A), A    ;出力クリア
LD      (CTRL_A_SAVE), A ;出力データ保存

;      LED ONルーチン
LED_ON: PUSH    AF
LD      A, (CTRL_A_SAVE)
OR      1           ;bit 1 ON
OUT     (CTRL_A), A    ;LED ON
LD      (CTRL_A_SAVE), A ;保存
POP     AF
RET

;      LED OFFルーチン
LED_OFF: PUSH    AF
LD      A, (CTRL_A_SAVE)
AND     0FEH        ;bit 1 OFF
OUT     (CTRL_A), A    ;LED OFF
LD      (CTRL_A_SAVE), A ;保存
POP     AF
RET
    
```

という意味をもたせたとします。いま LED が ON、モータが OFF している状態で、モータを ON したいとします。単にビット 7 を 1 にして 80h をポートに出力したらどうなるでしょうか。ビット 0 まで OFF になり、LED が消えてしまいます。

このようなときのために、出力専用ポートに出力したデータを保存しておき、操作したいビットに対してのみ ON/OFF 操作をしたあとそのデータを出力すれば、ほかのビットの状態に影響を与えないでビットのコントロールを行うことができます(リスト 1)。

#### ● 同一アドレスに入出力ポートを割り当てる

$\overline{Y_0}$  と  $\overline{IOWR}$  を AND した信号を HC273 の CK 端子へ、そして  $\overline{Y_0}$  と  $\overline{IORD}$  を AND した信号を HC245 の  $\overline{G}$  へ接続するとどうなるでしょう(図 13)。つまり入力と出力を同一 I/O アドレスに割り振るのです。

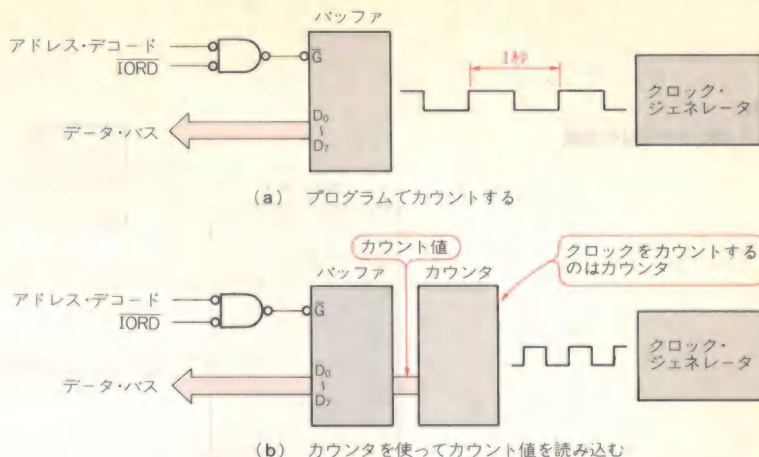
この I/O アドレスに対して出力動作をすると、データ・バスのデータが HC273 に書き込まれます。また入力動作をすると、データ・バスに HC245 のデータが出力されます。

ある一つの装置のコントロールのために、書き込みと読み出しの必要があるときは、このように同じアドレスに読み込みと書き込みを割り付けるとわかりやすくなります。

また場合によっては I/O アドレス・デコード回路の



〈図 14〉  
時計機能を実現する



ゲート数節約のために、まったく別の意味をもつ I/O デバイスのアドレスを共有して、出力ポートと入力ポートを同一アドレスにすることもあります。

同じアドレスでありながら、出力は装置 A のコントロールの機能を、入力では装置 B のステータスを読み込むという使い方も可能です。

また HC273 の  $Q_1 \sim Q_8$  の出力を HC245 の  $B_1 \sim B_8$  に接続すると、出力で HC273 に書き込んだデータを HC245 から読み込むことが可能になります。これでワーク・エリアにいちいちデータを保存しなくても、出力したデータを読み込むことができます。

## I/O コントローラとは何か

### ● ラッチやバッファ以外の I/O デバイス

いままでは非常に簡単なラッチやバッファによる I/O 回路について説明しました。しかし実際にマイコンをコントローラとして使う場合は、**接続する外部装置の形態や扱うデータの形式によって、それぞれ特化した I/O デバイスがあると便利です。**

例えばマイコンに時計機能を実現させるために、いままで説明してきたバッファでこれを作るとすると、1 秒周期のクロックをバッファから読み込んで、プログラムでカウントしなければなりません [図 14 (a)]。

もう少し頭をひねり汎用ロジック IC のカウンタを使い、そのカウント・データをバッファを介して読み込めば、プログラムでクロックをカウントする必要はありません [図 14 (b)]。

図をみると最低バッファとカウンタが必要で、カウンタの値をクリアしたいなどの機能を入れるとさらに回路は複雑になります。これらクロックのカウント機能を一つのチップに詰め込んだ LSI があると非常に便利です。このクロックのカウントに特化した LSI が CTC と呼ばれる LSI なのです。

〈表 2〉 Z80 ファミリの種類

PIO (パラレル出力コントローラ)
パラレルで外部装置とやりとりをする基本的な I/O。A-D コンバータの制御や EPROM ライタの制御によく使われる。
SIO (シリアル入出力コントローラ)
シリアルで外部装置とやりとりをする I/O。RS-232-C のコントローラとして使われる。
CTC (カウンタ・タイマ・コントローラ)
時間やクロックを数えるカウンタ。数を数えるものに使われる。
DMA (ダイレクト・メモリ・アクセス・コントローラ)
メモリや I/O 間で CPU を介さずに直接データの転送を行うためのコントローラ。最近の Z80 システムでは使われなくなってきた。

### ● マイコンで汎用的に使われるコントローラ

このカウンタ・タイマ用のコントローラである **CTC** (カウンタ・タイマ・コントローラ) 以外にも、ラッチやバッファの機能を一つで実現でき、プログラムによって出力や入力を設定できる **PIO** (パラレル I/O)、また RS-232-C などのシリアル通信用には **SIO** (シリアル I/O) があります。

これら CTC、PIO、SIO はマイコン・システムにおいては汎用的に使われる機能で、非常にたくさんの種類の LSI があり、それぞれに細かい機能の違いや特徴があります。

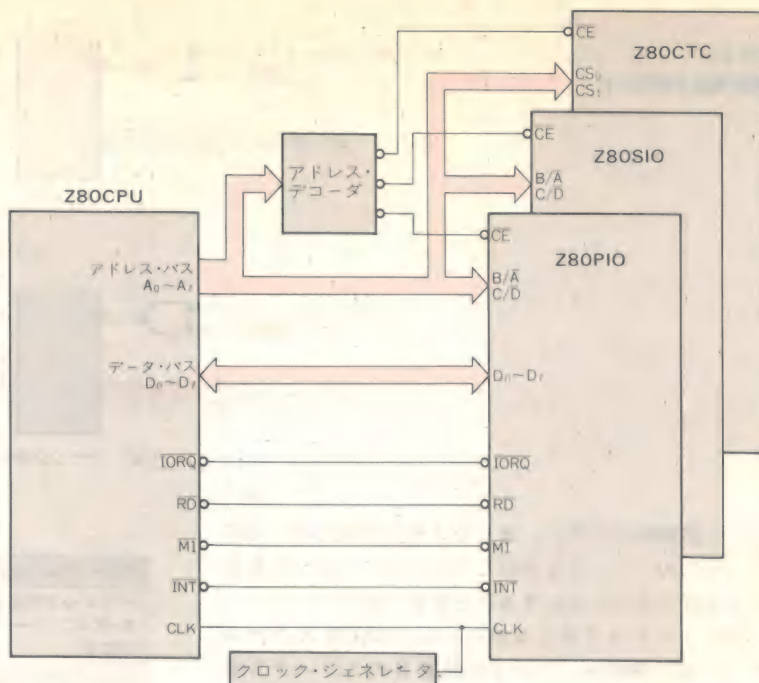
よって機能や使用する CPU との接続のしやすさなどから選択します。ここでは CPU として Z80 を取り上げているので、Z80 に接続しやすいコントローラということになります。

## Z80 ファミリ LSI の接続

### ● Z80 ファミリ LSI とは

Z80 の特徴の一つに、Z80 用周辺コントローラ LSI

## Z80 と Z80 ファミリの接続



の充実があります。これには **Z80PIO**(パラレル I/O)、**Z80CTC**(カウンタ・タイマ)、**Z80SIO**(シリアル I/O) があります(表 2)。これらはさすが Z80 ファミリーを名乗っているだけあって、**Z80CPU** と非常に簡単に接続することができます。

また後述するモード2割り込みを使うための仕掛けが、これらファミリ LSI には内蔵されているので、複雑で強力な Z80 のモード2割り込みシステムを、周辺 LSI を接続するだけで実現できます。

## ● Z80 ファミリと Z80CPU の接続

・Z80 周辺 LSI は、基本的には Z80CPU 専用で作られている周辺コントロール LSI 群です。当然 CPU と簡単に接続できるように考えられています。

図 15 に Z80 ファミリを接続するときの概念図を示します。Z80 ファミリは CPU と同期を取るために

CPU と同じシステム・クロックを必要とします。また  $\overline{\text{IORQ}}$ 、 $\overline{\text{RD}}$ 、 $\overline{\text{M1}}$ 、 $\overline{\text{INT}}$  の各制御線を接続します。

ラッチやバッファの例では、 $\overline{\text{IORQ}}$  と  $\overline{\text{RD}}$  を AND した  $\overline{\text{IORQ}}$  などの信号を使いましたが、Z80 周辺 LSI では、直接  $\overline{\text{IORQ}}$  を接続する端子があるので、必ず I/O 空間に対するアクセスで選択されるようになっていきます。

●  $\overline{\text{WR}}$  端子は接続しない

Z80 周辺・LSI は  $\overline{\text{WR}}$  端子をもっておりません。これでは書き込みができないように思えます。

しかしタイミングをよく考えてみれば、 $\overline{\text{RD}}$  と  $\overline{\text{WR}}$  が同時にアクティブになることはあり得ません。そこで、 $\overline{\text{IORQ}}$  がアクティブになったとき、 $\overline{\text{RD}}$  が L レベルなら読み込み、H レベルなら書き込みであると判断しているのです。

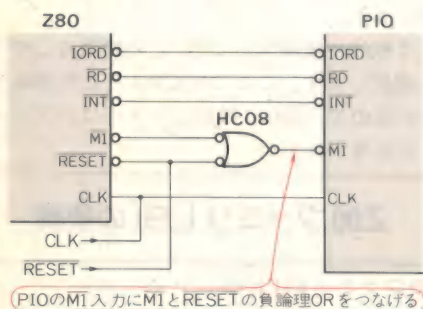
このように Z80 ファミリーは、ピン数の都合から内部で  $\overline{\text{WR}}$  を作っているのですが、 $\overline{\text{WR}}$  信号を接続しなくても大丈夫なのです。ただし例外的に Z80DMA には  $\overline{\text{WR}}$  端子があります。

### ● $\overline{M1}$ の重要性

M1 も Z80 周辺 LSI にとって非常に重要な線です。Z80 周辺 LSI は常時データ・バスを監視して、CPU が割り込み処理からメイン・ルーチンに戻るときの RETI 命令を実行するかどうかをチェックしているからです。

また CPU が割り込みベクタを読み込むときに、 $\overline{M1}$  と  $\overline{IORQ}$  をアクティブにします。周辺 LSI はこ

〈図 16〉 Z80PIO のリセットのかけ方





の信号が両方 L レベルになったとき割り込みベクタを出力します。割り込みに関する詳しい解説は第 6 章を参照してください。

### ● Z80PIO のリセット

また PIO だけは 40 ピンのパッケージに収めるために、 $\overline{\text{RESET}}$  信号の入力端子を省略し、 $\overline{\text{M1}}$  ピンを  $\overline{\text{RESET}}$  信号と兼ねています。

Z80PIO は内部にパワー ON リセット回路を内蔵していますが、外部より強制リセットするときは、 $\overline{\text{IORQ}}$  と  $\overline{\text{RD}}$  を H レベルにして  $\overline{\text{M1}}$  信号を 2 クロック・サイクル以上 L レベルにします。 $\overline{\text{M1}}$  はオペコード・フェッチ時に L レベルになる端子で、第 1 章で説明したようにオペコード・フェッチ・サイクルは 4 クロックで、このうち  $\overline{\text{M1}}$  が出力されるのが 2 クロック以内です。

つまり 2 クロック以上 L レベルが続いたときは、オペコード・フェッチ・サイクルではないと判断できるわけです。

よって Z80PIO の  $\overline{\text{M1}}$  入力へは、CPU からの  $\overline{\text{M1}}$  とリセット回路からのリセット信号の負論理 OR をとった信号を入力します (図 16)。

### ● アドレス・デコードの処理

残る信号線はアドレス・バスの処理です。Z80 周辺 LSI にはチップ・セレクト制御として  $\overline{\text{CE}}$  端子があります。また LSI 内部にコマンド用やステータス用など複数のポートやレジスタがあります。これらの選択

のために、 $\overline{\text{B/A}}$ 、または  $\overline{\text{C/D}}$  などのアドレス線があります。

ここでは例として Z80PIO の接続についてみてみます。

I/O アドレスのデコードにはいままでの例と同様 HC138 などを使います。しかしどのアドレス線でデコードするかはちょっと考えなくてはなりません。

Z80PIO は 8 ビットの平行ル・ポートを 2 チャンネルもっています。このポートの選択のために  $\overline{\text{B/A}}$  というチャンネル選択ビットがあります。またそのポートを入力にするか出力にするかなどのコントロール用のレジスタ・ポートと、実際にデータを入出力するためのデータ・ポートを選択する  $\overline{\text{C/D}}$  があります。

つまり Z80PIO は一つの LSI で四つのアドレスを必要とするコントローラなのです。プログラムで制御する場合はこれら四つのアドレスが、チャンネル A のデータ・アドレス、チャンネル A のコントロール・アドレス、チャンネル B のデータ・アドレス、チャンネル B のコントロール・アドレスと順番に並んでいたほうがわかりやすいでしょう。

このようにアドレスをマッピングするためのデコード回路を図 17 に示します。この四つのアドレスを順番に並べるためには、 $\overline{\text{C/D}}$  に  $\text{A}_0$  を、 $\overline{\text{A/B}}$  に  $\text{A}_1$  を接続すれば OK です。そして  $\overline{\text{CE}}$  に HC138 のセレクト信号を接続します。この HC138 に入力するアドレス線と、出力  $\overline{\text{Y}}_0 \sim \overline{\text{Y}}_7$  のどれを  $\overline{\text{CE}}$  につなぐかによって、

## I/O アクセス時にウェイトを入れる回路

マイコンでは I/O デバイスのアクセス速度はメモリのそれと比較して遅いのが一般的です。システムの動作クロックを決めるとき、遅いデバイスに合わせるのというやり方もありますが、これでは命令の処理速度まで遅くなってしまいます。

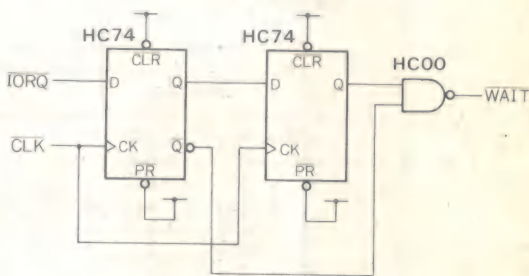
そこで、遅い I/O アクセスのときだけ CPU に「ちょっと待った!」をかけられる回路があると便利です。これがウェイト回路です。

簡単なウェイト回路例を図 A(a) に示します。この回路では I/O アクセスのときに 1 ウェイト入ります。

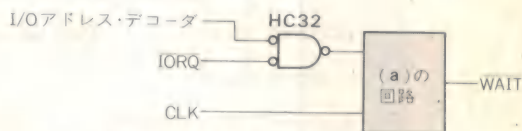
また特定の I/O だけにウェイトをかけたいときは、ウェイトをかけたいアドレスをアクセスしたときに L レベルになる信号、つまり I/O アドレス・デコードの出力信号を入力すれば、そのアドレスのアクセス時にだけウェイトを挿入できます (図 A(b))。

またさらに  $\overline{\text{WR}}$  信号も入力すれば、書き込みのときだけウェイトを入れるなどの処理も可能です。

〈図 A〉ウェイト挿入回路



(a) I/O サイクル時に 1 ウェイト入れる回路



(b) 特定の I/O アドレスだけにウェイトを入れる

**Z80CPU**

**Z80PIO**

**HC138**

**アドレス・バス**

**ポイント A**

**ポイント B**

**ほかのI/Oデバイスのセレクトへ**

**Z80ファミリではWRはいらない**

**Z80PIOだけ**

**Z80PIO以外のZ80ファミリ**

**Z80PIO以外はRESETをつなぐ**

[illegible]



Z80PIO の割り振られるアドレスが決まります。

図の例では  $A_5 \sim A_7$  も HC138 に入力しているので、Z80PIO のアドレスは  $20h$ 、 $21h$ 、 $22h$ 、 $23h$  の四つとなり、8 ビットをフル・デコードしていることになります。

## 80 系周辺 LSI の接続

### ● 8255A や 8251A をつなぐ

Z80 マイコンは 8080A の改良版として登場しましたが、Z80 周辺 LSI は当時としては斬新で使い方が難しかったため、従来からの 80 系周辺 LSI が依然として使われました。

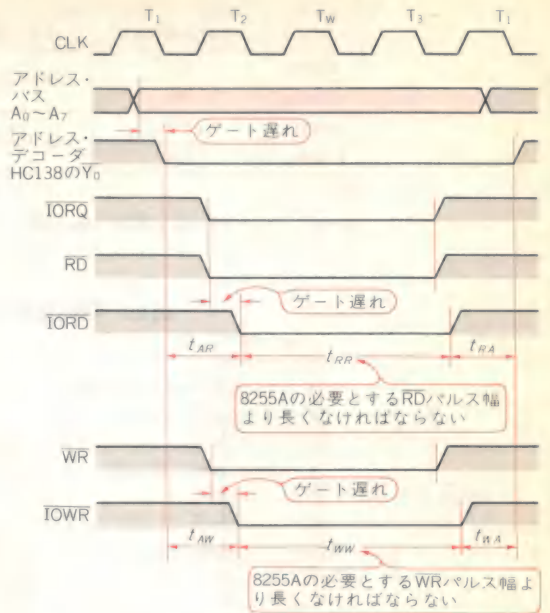
いまだに PPI(8255A)、USART(8251A)は、Z80 マイコン・システムでもお馴染みの周辺 LSI です。ちなみに、PPI や USART はパソコンでもよく使われています。

80 系周辺 LSI に対するアクセスは、メモリと同じように  $\overline{RD}$ 、 $\overline{WR}$ 、 $\overline{CS}$  信号で行います。また内部をクリアする  $\overline{RESET}$  信号は Z80 ファミリと極性が反対で、H レベルでアクティブです。

### ● Z80 と 80 系周辺 LSI の接続

80 系周辺 LSI として 8255A をとりあげ、図 18 に Z80 との接続例を示します。8255A にも Z80PIO と同様ポート選択やモード設定のためにアドレスがいくつかあります。そこでアドレス・バスの  $A_0$ 、 $A_1$  を接続し

〈図 19〉 Z80 と 8255A を接続したときのアクセス・タイミング



ます。

80 系周辺 LSI と Z80 周辺 LSI との一番の違いは、 $\overline{IORQ}$  信号の入力ピンがないことです。このため、 $\overline{RD}$ 、 $\overline{WR}$  両方に、もしくは  $\overline{CS}$  のどちらかに必ず  $\overline{IORQ}$  を接続しなければなりません。

## Z80 七不思議 I/O 編

**Q** : I/O空間のアドレスを16ビットにしなかったのはなぜ？

**A** : 8080では、I/O空間は256バイトなので、それに合わせてこの大きさになったと思われます。たしかに正式にサポートしている I/O 空間は 256 バイトしかありませんが、Z80CPU では I/O 命令を大幅に拡張し、間接 I/O アドレッシングや、I/O に対するブロック転送命令もサポートするようになりました。直接アドレッシングでは 8 ビットの I/O アドレスしか指定できませんが、拡張された命令の中には間接的に上位 8 ビットの部分を指定することができるものがあり、その命令を使うことにより 16 ビット、64K バイトの I/O 空間を使用することもできます。

もっとも Z80 の開発の時点では、256 バイト以上の I/O を必要とするアプリケーションがほとんどな

かった、というのが真実のような気がしますが。

**Q** : I/O命令時に自動的に1ウェイト入るのはなぜ？

**A** : Z80周辺デバイスは、その I/O サイクル、 $\overline{RETI}$  サイクル、割り込み応答サイクルを検出する必要があり、そのため CPU からの制御線をクロック・エッジでサンプルし、次のサイクル以降で動作をするようになっています。

Z80CPU では、ご存じのように  $\overline{IORQ}$  や  $\overline{RD}$ 、 $\overline{WR}$  は  $T_2$  の立ち上がりで変化しますので、通常のメモリ・サイクルと同じタイミングだとウェイトを入れないと苦しかったから、という理由からではないかと思います。また、Z80CPU が開発された当時の周辺デバイスのスピードがそんなに速くなかったこともあるでしょう。

〈信垣育司〉

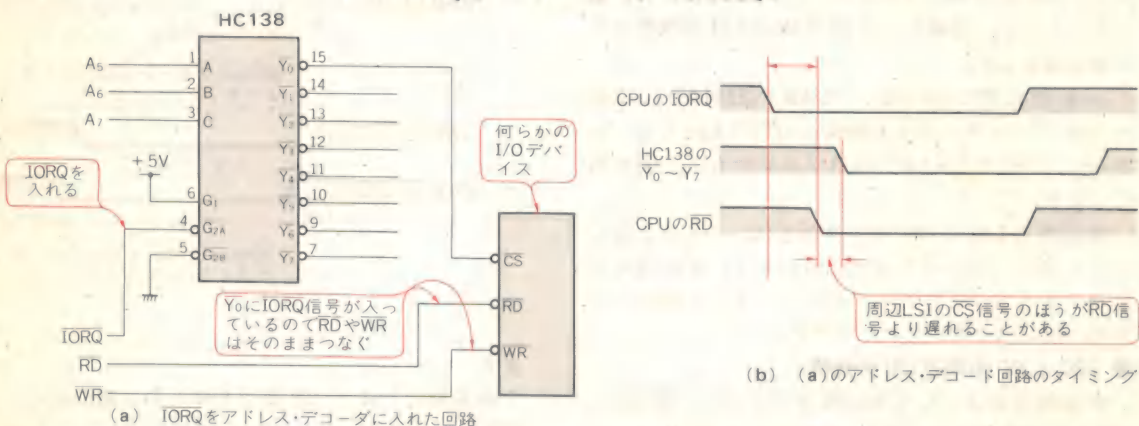
〈表3〉 Z80に8255Aを接続したときのアクセス・タイミングの計算

CPU クロック	Z80CPUのRD パルス幅( $t_{RR}$ )	8255Aの必要とする パルス幅( $t_{RR}$ )	Z80CPUのWR パルス幅( $t_{WW}$ )	8255Aの必要とする パルス幅( $t_{WW}$ )
6MHz	251	160(5MHz版)	256	120 (5M/10MHz版)
8MHz	188		193	
10MHz	150	150(10MHz版)	155	
12MHz	125		120	

単位: ns

注)  $t_{RR}' = t_{CK} \times 1.5 - t_{DCF}(\overline{RD}) + t_{DCF}(\overline{RD})$   $t_{WW}' = t_{CK} \times 1.5 - t_{DCF}(\overline{WR}) + t_{DCF}(\overline{WR})$

〈図20〉  $\overline{IORQ}$  をアドレス・デコードに入れたとき



回路例では  $\overline{IORQ}$  信号を  $\overline{RD}$  信号と  $\overline{WR}$  信号で負論理 AND した  $\overline{IORD}$  と  $\overline{IOWR}$  を、8255A の  $\overline{RD}$  と  $\overline{WR}$  に接続しています。通称インテル結線といいまします。また  $\overline{CS}$  には HC138 によるデコード出力を接続します。

### ● アクセス・タイミングの計算例

Z80 周辺 LSI と Z80CPU の接続は、同じ Z80 フタミリですから動作周波数の同じバージョンを使えば基本的には何も問題ありません。Z80CPU が 8 MHz 版であるなら、Z80PIO も 8 MHz 版を使います。動作クロックが異なるバージョン同士ではアクセスが追いつかず、問題があります。

アドレス・デコードも、HC138 などを普通に使う限りではまず問題ありません。よほど複雑で変なデコードをしないかぎり、ゲートの遅れはあまり気にしなくてもだいじょうぶです。

ここでは Z80CPU と 8255A などの 80 系周辺 LSI とを接続した場合です。

I/O ポートからのデータ入力時は  $\overline{CS}$  と  $\overline{RD}$  信号が、データ出力時は  $\overline{CS}$  と  $\overline{WR}$  信号がアクティブになります。ただしそのタイミングに注意します。

一番問題になるのは、8255A の必要とする  $\overline{RD}$  信号と、 $\overline{WR}$  信号のパルス幅です。Z80 と 8255A をつな

げたときの  $\overline{RD}$  信号と  $\overline{WR}$  信号のパルス幅との関係をまとめると、図 19 と表 3 のようになります。表を見ると、12 MHz の Z80 では 10 MHz 動作の 8255A の  $\overline{RD}$  パルス幅を満足できないことがわかります。

ここでアドレス・デコードを図 20 のような回路にして、 $\overline{CS}$  に  $\overline{IORQ}$  を入れた場合を考えてみます。HC138 のゲート遅れによって、 $\overline{RD}$  や  $\overline{WR}$  より  $\overline{CS}$  が遅れることが考えられます。このようなデコードによる回路がたまたに見受けられますが、正確にはタイミングに問題があります。

$\overline{CS}$  でイネーブルをかけてから  $\overline{RD}$  や  $\overline{WR}$  で読み書きするタイプの I/O デバイスでは、この点に注意する必要があります。

ただし、デバイスの中には  $\mu$ PD71055(NEC) のように、 $\overline{CS}$  より  $\overline{RD}$  が先に立ち下がっても動作を保証しているものもあります。

### ● 参考文献

- (1) 鈴木八十二, 村田浩義: CMOS マイコン Z80 を用いたシステム設計, 日刊工業新聞社。

(トランジスタ技術 1994 年 6 月号に加筆, 修正)



電源ONからマイコンを動作させるまでの回路

クロック&リセット周辺回路の設計

野口智樹

Z80 の電源とクロック回路

この章では Z80 を動作させるために必要な電源やクロック、そしてマイコンの動作のすべてのスタートであるリセット回路について説明します。

● Z80 の電源

表 1 に Z80 の電源特性を示します。CMOS 版では HALT 命令の実行とクロックを停止させることで、スタンバイ・モードと呼ばれる低消費電力モードを備えています。このときの消費電流は 10  $\mu$ A と非常に

〈表 1〉 CMOS 版 Z80 の電源特性

記号	名 称	最小	最大	条 件	単位
$V_{CC}$	電源電圧	+4.50	+5.50		V
$V_{IL}$	入力 “L” 電圧	-0.3	0.8		V
$V_{IH}$	入力 “H” 電圧	2.2	$V_{CC}$		V
$V_{OL}$	出力 “L” 電圧		0.4	$I_{OL}=2.0\text{mA}$	V
$V_{OH1}$	出力 “H” 電圧	2.4		$I_{OH}=-1.6\text{mA}$	V
$V_{OH2}$	出力 “H” 電圧	$V_{CC}-0.8$		$I_{OH}=-250\mu\text{A}$	V
$I_{CC1}$	消費電流	4MHz 動作	20	$V_{CC}=5\text{V}$ $V_W=V_{CC}-0.2\text{V}$ $V_{IL}=0.2\text{V}$	mA
		6MHz 動作	30		mA
		8MHz 動作	40		mA
		10MHz 動作	50		mA
		20MHz 動作	100		mA
$I_{CC2}$	スタンバイ時消費電流		10	CLK=0	$\mu\text{A}$

小さく、電池動作の場合などに非常に有利になります。

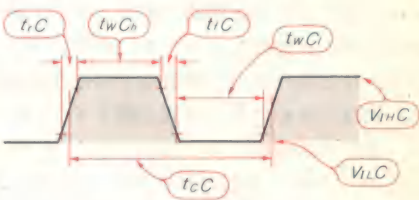
● 基本的なクロック発生回路

図 1 に Z80 のクロック入力波形を、表 2 にクロック入力特性を示します。また図 2 にいちばん基本的なクロック発生回路を示します。使用した水晶振動子の周波数でクロックが作られます。

表 2 をみると、クロックの H レベルと L レベルは同じ時間となっています。つまりデューティ比が 1:1 のクロックが必要だということです。図 2 (a) の回路では場合によってはデューティ比 1:1 の波形が得られないことがあるので、図(b)のように間にフリップフロップを入れて 1/2 に分周し、正確なデューティ比 1:1 のクロックを得るようにします。当然この場合は、水晶振動子の発振周波数を 2 倍にしないと、図(a)と同じクロック周波数が Z80 に供給されません。

図 3 に実用的な発振回路を示します。図(a)の回路

〈図 1〉 Z80 のクロック入力波形

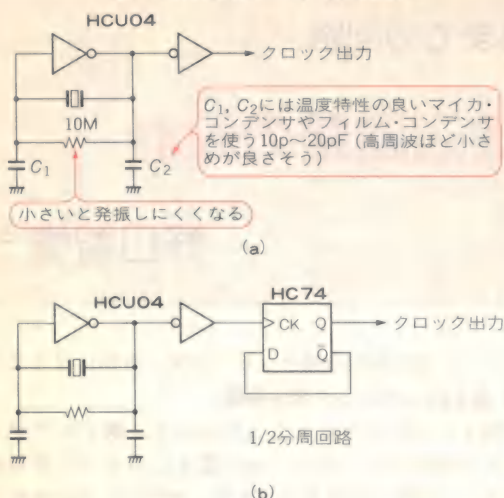


〈表 2〉 CMOS 版 Z80 のクロック入力特性

記号	項 目	6MHz 版		8MHz 版		10MHz 版		20MHz 版		単位
		最小	最大	最小	最大	最小	最大	最小	最大	
$T_{CC}$	クロック周期	162	DC	125	DC	100	DC	50	DC	ns
$t_{wCh}$	クロック “H” パルス幅	65	DC	55	DC	40	DC	20	DC	ns
$t_{wCl}$	クロック “L” パルス幅	65	DC	55	DC	40	DC	20	DC	ns
$t_{rC}$	クロック立ち上がり時間		20		10		10		10	ns
$t_{fC}$	クロック立ち下がり時間		20		10		10		10	ns

記号	項 目	CMOS 版 Z80 共通		単位
		最 小	最 大	
$C_{CLOCK}$	クロック入力端子静電容量		10	pF
$V_{ILC}$	クロック入力 “L” 電圧	-0.3	0.45	V
$V_{IHC}$	クロック入力 “H” 電圧	$V_{CC}-0.6$	$V_{CC}+0.3$	V

〈図2〉 基本的なクロック発生回路



をみると出力にプルアップ抵抗がついています。これは表2をみるとわかるように、**クロック入力のHレベルの電圧が  $V_{cc} - 0.6V$** であるためです。しかしこうするとクロックの立ち上がり、立ち下がり時間などに影響が出るため、本当は好ましくありません。

CMOSのインバータではプルアップの必要はありません。

#### ● 部品や実装上の注意点

回路中のコンデンサには温度特性のよいマイカ・コンデンサやフィルム・コンデンサを使います。容量は10 p~20 pF程度ですが、クロック周波数が高い場合は小さめの容量がよいようです。

マイコンはデジタル回路ですが、この**クロック発生回路はアナログ動作**といってよいので、プリント基板化のときなどはパターン配置などに注意が必要です。

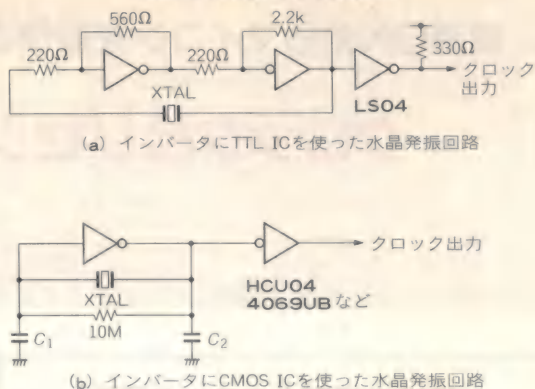
パターンは小さめにして可能な限りインバータのすぐ隣に、また**水晶振動子の付近はベタ・アース**で他の信号などのパターンを引き回さないようにします(図4)。

また発振回路に使用するインバータの数は2個程度なので、汎用ロジックICを使うと必ずゲートが余ります。このゲートも何かに使いたくなるでしょうが、できれば何にも使用せずに**入力をグラウンドに落としたほうがよい**でしょう。余ったゲートを使った場合、そのゲートのスイッチング動作が、発振周波数の精度に影響を与えます。

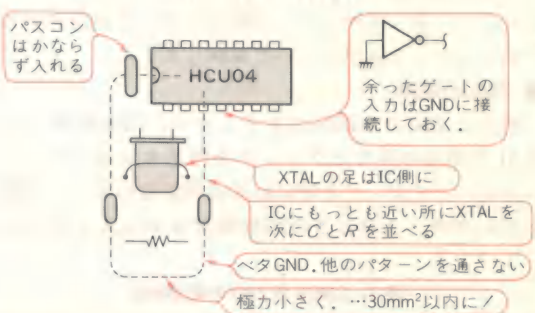
水晶振動子は熱に弱いので部品取り付けの順番に注意します。

またこの回路で発生できるクロック周波数は、最高十数MHz程度と考えてください。20MHz以上のクロック周波数が必要な場合は、専用のクロック・ジェネレータを使うのが賢明です。

〈図3〉 実用的な水晶発振回路



〈図4〉 実装上の注意点



#### ● 専用クロック・ジェネレータを使った回路

最近では水晶発振子と分周器を内蔵した専用の**クロック・ジェネレータIC**が多数発売されています。図2で使用した水晶発振子とインバータによる発振回路より信頼性があり、クロックの精度も非常に高いのが特徴です。

クロック・ジェネレータにはEXO3シリーズ(キンセキ)がよく使われます。これは指定クロックの発振のほかに、これを分周したクロックも出力できます。

図5にピン配置と分周比の設定を示します。また表3にEXO3が対応している周波数の一覧を示します。

#### ● クロック周波数の選び方

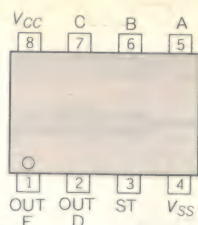
クロック回路の設計についてはわかりましたが、では具体的にクロック周波数はどう選択すればよいのでしょうか。

当然、使用するZ80の最高クロックより高速なクロックは使用できません。4MHz版のZ80なら最高4MHzです。

この最高周波数を越えなければ、基本的にどんな周波数でもかまいません。4MHz版だからといって必ず4MHzのクロックで使わなければならない規則はどこにもありません。また**CMOS版**であれば最低クロックはDC、つまり**完全にクロックを停止すること**



〈図 5〉 EX03 のピン配置と分周比



(a) ピン配置

入 力				出 力	
選 択				F(原振 周波数)	D (分周波)
C	B	A	ST		
X	X	X	L	L	L
L	L	L	H	$f_o$	$f_o/2$
L	L	H	H	$f_o$	$f_o/2^2$
L	H	L	H	$f_o$	$f_o/2^3$
L	H	H	H	$f_o$	$f_o/2^4$
H	L	L	H	$f_o$	$f_o/2^5$
H	L	H	H	$f_o$	$f_o/2^6$
H	H	L	H	$f_o$	$f_o/2^7$
H	H	H	H	$f_o$	$f_o/2^8$

H: ハイ・レベル,

L: ロー・レベル,

X: Don't care

(b) 分周比

も可能です。

NMOS 版では、CPU の内部動作が人間が自転車に乗って走っている状態のような、ダイナミック動作で動いているため、常にクロックを入力しないと動作できません。つまり最低クロック周波数というのが決まっているので、これより遅い周波数は選択できません。

NOP 命令 1 個 2 個でクロック時間を計算する必要があるなら、4 MHz や 10 MHz といった切りのよい周波数のほうが実行時間を計算しやすいでしょう。

また、この CPU のクロックを SIO の通信用の送受信クロックにも使うような設計も多くみられます。この場合はその通信用のクロックを分周して作り出すのに都合のよい周波数を選択すべきです。

表 4 に通信用クロックの分周に最適なマスタ・クロック周波数を示します。6 MHz 付近では 6.144 MHz と、6 MHz を少し越えています。これはもともと 6.144 MHz での使用を意識していて、6 MHz 版の

〈表 4〉 通信用クロックの分周に最適なクロック周波数の例

CPU クロック	システム・クロック
5MHz 付近	4.915MHz
6MHz 付近	6.144MHz
8MHz 付近	7.987MHz
10MHz 付近	9.83MHz
16MHz 付近	15.974MHz
20MHz 付近	19.66MHz

Z80 は最高動作クロックが 6.17 MHz になっているので大丈夫です。

## Z80 のリセット回路

### ● 基本的なリセット回路

図 6 に Z80 のリセット・クロック入力波形のようすと、表 5 にリセット・サイクルを示します。これを見るとわかるように、Z80 は入力されるクロックで 3 クロック分の間、リセット端子を L レベルにしなればなりません。

図 7 に抵抗とコンデンサによるもっとも基本的なリセット回路と、各部の波形の変化のようすを示します。

リセット時間は  $C$  と  $R$  による時定数で決まります。この回路では、

$$\begin{aligned} \text{リセット時間} &= 10 \times 10^3 \times 10 \times 10^{-6} \times 0.66 \\ &= 0.066 \text{ [s]} \end{aligned}$$

となります。

この時間が Z80CPU をリセットするために必要な、3 クロック分の時間かどうかはもはや計算する必要ありません。1 MHz 動作の Z80 でも十分すぎるほどの時間です。

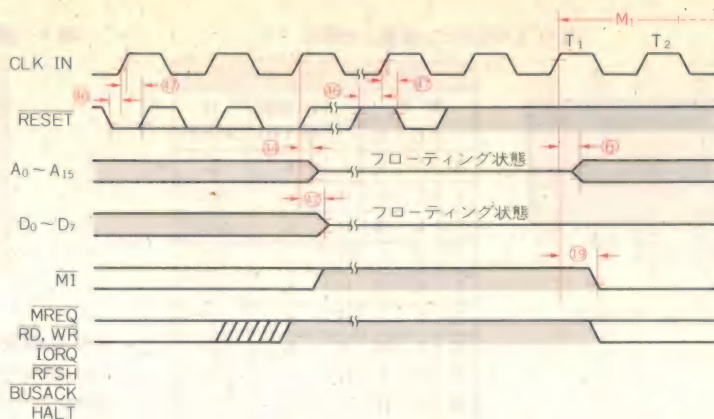
また間にシュミット・トリガ入力のインバータが 2

〈表 3〉  
EX03 の周波数

原発振	分 周 出 力							
$1/2^0$	$1/2$	$1/2^2$	$1/2^3$	$1/2^4$	$1/2^5$	$1/2^6$	$1/2^7$	$1/2^8$
MHz				kHz				
12	6	3	1.5	750	375	187.5	93.75	46.875
12.288	6.144	3.072	1.536	768	384	192	96	48
12.8	6.4	3.2	1.6	800	400	200	100	50
14.31818	7.15909	3.579545	1.789772	894.88	447.44	223.72	111.86	55.930
14.7456	7.3728	3.6864	1.8432	921.6	460.8	230.4	115.2	57.6
15.9744	7.9872	3.9936	1.9968	998.4	499.2	249.6	124.8	62.4
16	8	4	2	1000	500	250	125	62.5
16.384	8.192	4.096	2.048	1024	512	256	128	64
17.73447	8.867238	4.433619	2.216809	1108.40	544.20	277.10	138.55	69.275
18.432	9.216	4.608	2.304	1152	576	288	144	72
19.6608	9.8304	4.9152	2.4576	1228.8	614.4	307.2	153.6	76.8
20	10	5	2.5	1250	625	312.5	156.25	78.125
24	12	6	3	1500	750	375	187.5	93.75

〈図6〉

Z80のリセット・サイクルのタイミング



〈表5〉 Z80のリセット・サイクル(単位 ns)

番号	記号	項目	6 MHz 版		8 MHz 版		10 MHz 版		20 MHz 版	
			最小	最大	最小	最大	最小	最大	最小	最大
6	$t_{dCF(A)}$	クロック立ち上がりからの有効アドレス出力遅延		90		80		65		57
19	$t_{dCF(MI)}$	クロック立ち上がりから $MI = "L"$ になるまでの遅延		80		70		65		45
42	$t_{dCF(DZ)}$	クロック立ち上がりからデータバス・フロート状態までの遅延		80		70		65		40
44	$t_{dCF(AZ)}$	クロック立ち上がりからアドレスバス・フロート状態までの遅延		80		70		75		40
46	$T_{sRESET(CR)}$	クロック立ち上がりに対する $\overline{RESET}$ セットアップ時間	60		45		40		15	
47	$T_{hRESET(CR)}$	クロック立ち上がりから $\overline{RESET}$ ホールド時間	10		10		10		10	

段入っているのは、Z80 のリセット入力端子にヒステリシス特性がないためです。図のような徐々に電圧が上昇する信号を直接リセット端子には入力できません。

また場合によっては正論理でリセットをかける周辺 LSI を使うかもしれません。このときは図のように 1 段目と 2 段目の間からリセット信号を引くとよいでしょう。

### ● リセット時間の注意点

Z80 は確かに 3 クロック分の時間でリセットをかけられますが、この Z80 のクロックに、専用のクロック・ジェネレータ IC を使う場合は注意が必要です。

クロック・ジェネレータの中には、電源投入後数百 ms ものクロック発振準備時間を必要とするものもあります。これらの準備時間も考えて、リセット時間を決めなければなりません。

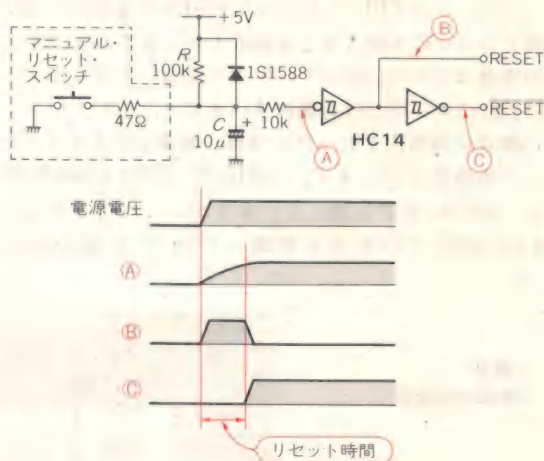
### ● CR によるリセット回路の問題点

この CR によるリセット回路は基本的で簡単な回路なので多用されていますが、実は問題点もあります。

例えば、このリセット回路を搭載した CPU カードで、何らかの外部機器をコントロールするとします。電源投入の順序を外部機器から先に入れた場合、インターフェース部分のドライバに使用した IC の内部構造によっては、図 8 のように電流が CPU 側にも流れ、リセット用のコンデンサが充電されてしまいます。

このようにコンデンサが充電されたまま CPU 側の

〈図7〉 CR によるリセット回路



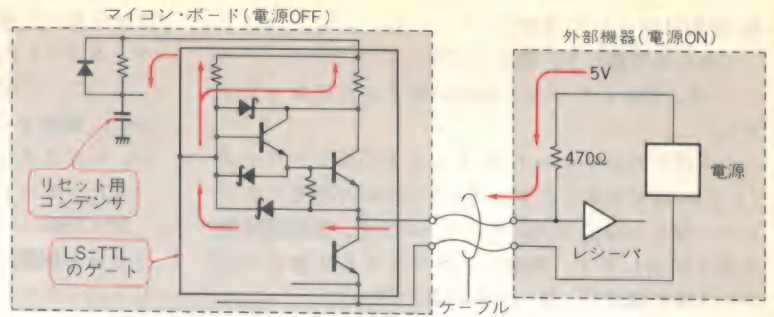
電源を入れたら、すでにコンデンサには電荷がたまっているわけですから、時定数の時間よりはるかに短い時間で充電が完了してしまい、L レベルの時間が短くなってしまいます。これでは CPU を正常にリセットできなくなってしまいます。

また電源電圧の立ち上がりがゆっくりな場合も正常にリセットできません。

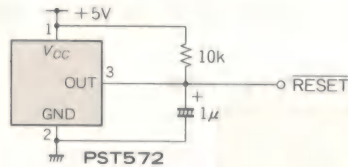
さらに電源装置に出力電圧の不安定なものを使って



〈図 8〉  
抵抗とコンデンサによる  
リセット回路の不都合の例

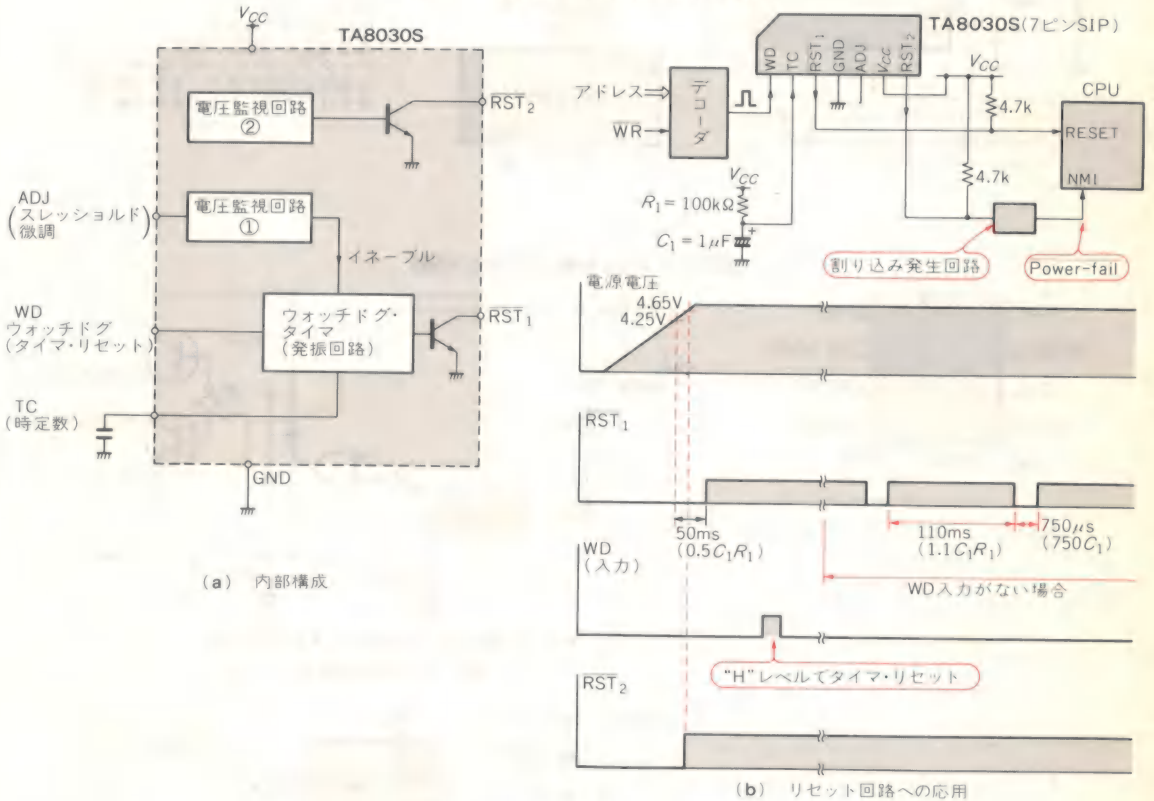


〈図 9〉  
PST572 を使った  
リセット回路



- ・ほかにPST518/520/523/524などもある
- ・PST573は正論理RESET出力タイプ
- ・PST572/573ともに小型面実装パッケージ (MMP3P) もある

〈図 10〉 TA8030S を使ったリセット回路



いる場合、CMOS の Z80 の場合 4.5 V が最低動作電圧ですから、たとえ瞬間的にでもこの電圧を下回ったら CPU の内部動作は保証できませんから、リセットをかけなければなりません。

しかし CR による回路では、すでに説明したように

波形整形のためにシュミット・トリガ入力のインバータを使います。

このヒステリシス特性のために、一度 H レベルになった信号は約 1 V 程度まで電圧が下がらないと L レベルになってくれません。

## ● 専用リセット IC を使う

これらを考慮して、最近では**電源監視やリセット用の IC**を、Z80 のリセット制御に使うようになってきました。

よく使われるリセット IC として PST572 シリーズ (ミツミ) などがあります。パッケージは小信号トランジスタなどと同じ TO92 パッケージです。使用回路例を図 9 に示します。電源とグラウンドとリセット出力の 3 端子構成で、使いやすい IC です。

また TA8030S (東芝) を使った回路例を図 10 に示します。この IC は装置全体にリセットをかける目的の電圧監視回路と、電圧が低下してきたことをチェッ

クする電圧監視回路をもっています。またウォッチドグ・タイマも内蔵しています。

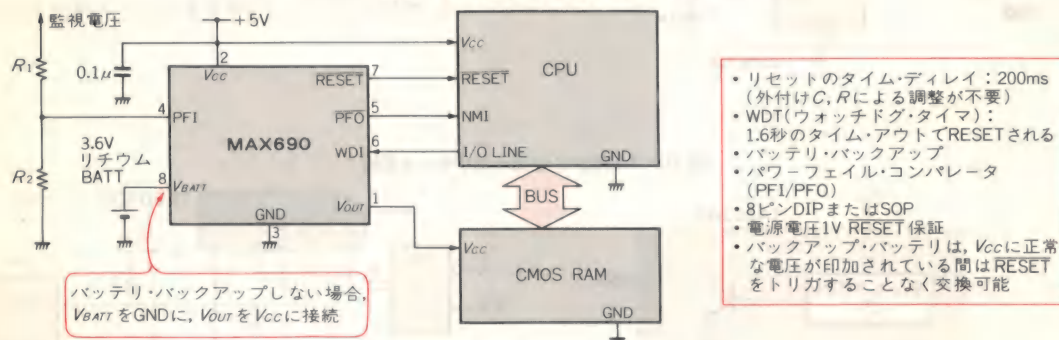
この二つの電圧検出を応用するとリセット制御のほかに、電源を OFF する瞬間に CPU に割り込みをかけ、レジスタなどの退避や保存をすることができます。

またリセット時間は外付けの CR で決められます。

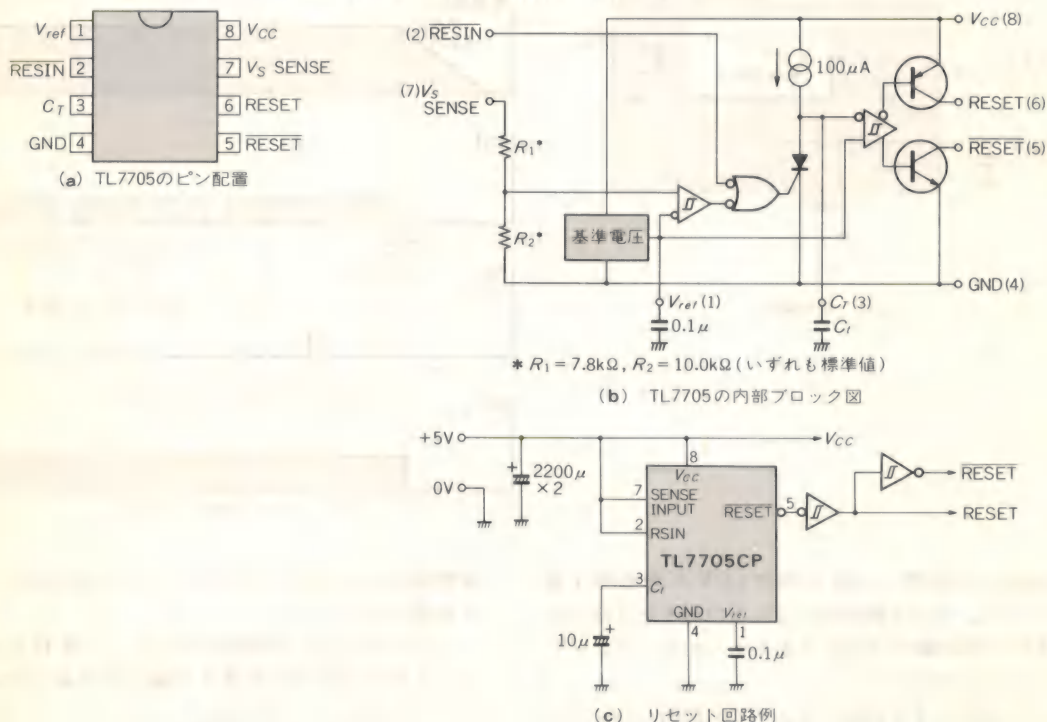
MAX690 シリーズ (マキシム) も高性能な IC です。リセット制御、ウォッチドグ・タイマ、そしてバッテリー・バックアップの制御機能をもっている点が特徴的です。図 11 に回路例を示します。

ほかに図 12 に TL7705 (テキサス・インスツルメンツ) というリセット IC のピン配置やブロック図を、最

〈図 11〉 MAX690 を使ったリセット回路

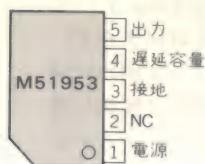
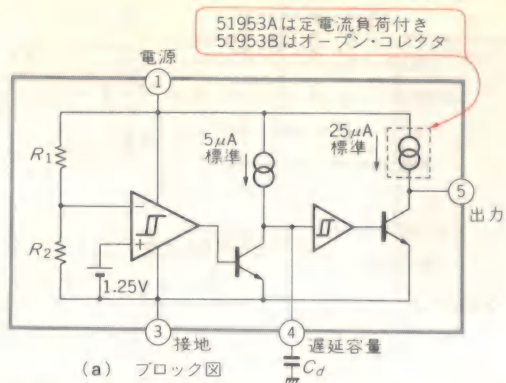


〈図 12〉 TL7705 を使ったリセット回路

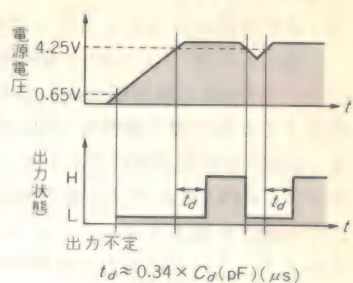




〈図 13〉 M51953B の内部構造とピン配置

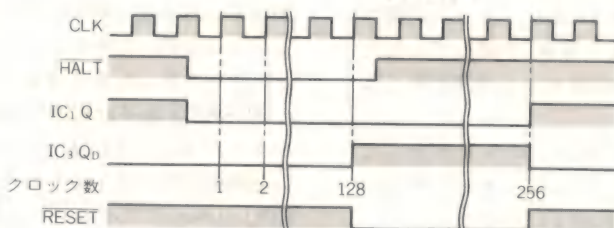
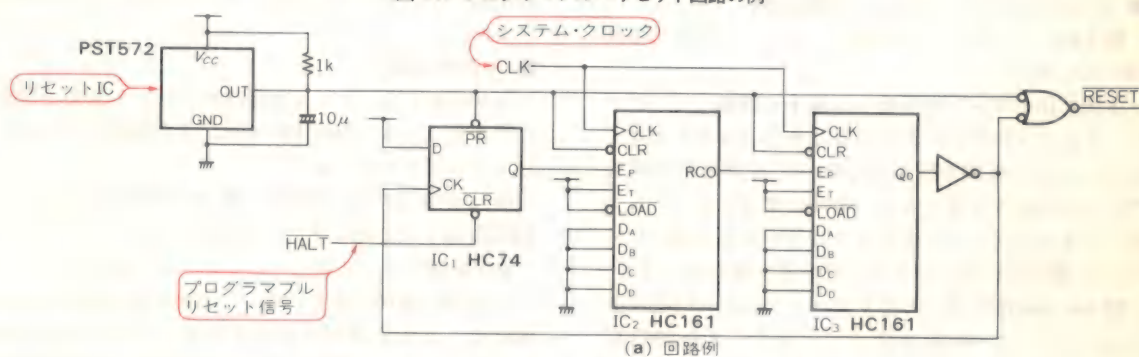


(b) ピン接続図



(c) 応答

〈図 14〉 プログラマブル・リセット回路の例



(b) タイミング

後に M51953(三菱電機)のピン配置やブロック図を図 13 にします。

● Z80のリセットとは

ここまで Z80 のリセットは、3 クロック分の時間リセット入力端子を L レベルにすると説明してきましたが、もっと正確にいうと、「リセット入力端子を L レベルにして、クロック入力端子に 3 クロック以上クロックを入れるとリセットがかけられる」といい直しましょう。

つまりクロックが正常に入力されていない状態では、いくら数百 ms、いや1秒以上リセット入力端子をLレベルにしたところで、Z80はまったくリセットされないことに注意してください。

## ● プログラマブル・リセットとは

メモリのパリティ・エラーなど、システムで何らかのエラーが検出された場合、その後の処理をどうするかは、システム・デザイン的な問題なので、その機器の用途や動作環境、要求される信頼性などから検討する必要があります。そのあたりの解説は、他のマイコン・システム設計などの書籍を参考にしてください。

ここでは、エラーを検出した後の処理や、リセット・スタートと同様にもう一度初めからシステムを動作させたい場合など、ソフトウェアから自分自身をリセットしたいときの方法について説明します。

リセットをソフトウェア的な面だけから見ると、割り込みモードが0および割り込み受け付け禁止状態で、

JP 0000H や、RST 00H など、アドレス 0000h 番地から実行を開始することと同じです。

しかしハードウェア的に **RESET** 信号は、**すべての順序回路** (出力値が現在の入力だけでは決まらず、直前までの出力値と現在の入力の値によって出力が決まる回路) を初期状態に戻すための、非常に重要な信号です。CPU およびその周辺回路は出力レジスタなどに代表されるように、フリップフロップが多用されています。これらの出力を初期状態に戻すための信号として、**RESET** 信号が使われるわけです。

よって、**RESET** 信号によって出力レジスタなどがハードウェア的に初期化される回路が使われているシステムの場合、たんに 0000h 番地から実行を再開しても、リセット・スタートと同様な動作とはなりません。

### ● プログラマブル・リセット回路の例

図 14 に、ソフトウェア的にリセットを発生する回路を示します。

単純に出力ポートをリセット端子に接続して、L レベルを出力すればよいかというとそうではありません。まず、本来の電源 ON からの動作でも正常に初期化されなければなりませんし、また正常なリセット入力には、システム・クロックで 3 クロック以上の間、L レベルを **RESET** 端子に入力し続ける必要があります。

図 14 の回路では、この 3 クロック以上の時間のカウントにカウンタ IC を使っています。また周辺 IC によっては 3 クロックよりさらに長いリセット時間が必要な場合があります。このような IC を使っているときは、カウント値を多くすればリセット期間が長くなります。

図ではプログラマブル・リセットのトリガとして **HALT** 信号を使っています。よってリセットをかけるには **HALT** 命令を実行すればよいわけです。

リセット IC によるリセット出力をハードウェア・リセット、図 14 のカウンタ出力によるリセット出力をソフトウェア・リセットと呼びます。CPU やプログラムのにもリセットをかけたいデバイスのリセット入力には、この二つのリセット信号を OR した信号を入力し、両方でリセットがかかるようにします。

ポート出力によってリセットをかけたい場合は、**HALT** の代わりにそのポートの任意のビットを接続します。ただし、その出力ポートはハードウェア・リセット時に H レベルで初期化される必要があります。さもないと、ハードウェア・リセットとソフトウェア・リセットの両方がかかりっぱなしという状態になりかねません。

〈図 15〉 その他の信号線の処理

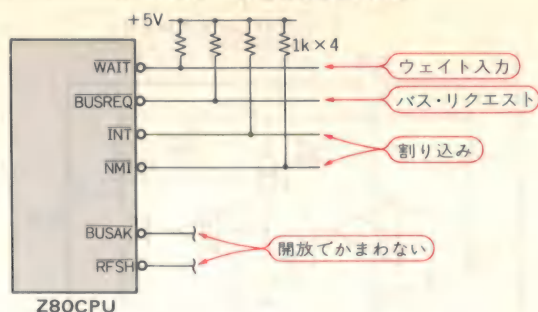


図 14 の回路では **HALT** 命令実行後、**HALT** 端子が L レベルになって 128 クロック目から **RESET** 出力が L レベルになり、128 クロックの間リセットをかけます。

### ● その他の端子

メモリや I/O、クロック回路やリセット回路を説明してきましたが、Z80CPU には、まだ説明していないピンがいくつかあります。

使っていない出力端子、例えば **BUSAK** 端子や **RFSH** 端子などは、そのまま開放でかまいません。

必ず処理しなければならないのは入力端子です。

まず **BUSREQ** 端子です。この端子は DMA などを接続するときに使用する端子です。メモリや I/O 制御するのは本来 CPU だけのはずですが、しかし場合によっては CPU 以外のコントローラがバスを制御するというシステムもあり得ます。

**BUSREQ** 端子は CPU 以外にバスを制御したいデバイスがあるときに L レベルを入力し、CPU にバスの制御権を明け渡してもらうための要求信号線です。CPU 以外のデバイスがバスを制御することがない場合は電源に接続してかまいませんが、一般的にはプルアップ抵抗をつけて H レベルにしておきます。

**WAIT** 端子は第 4 章のコラムで少し触れたように、アクセス速度が遅いデバイスが CPU に待たされるための信号線です。これも普通はプルアップしておきます。

**INT** と **NMI** は、次の割り込みの章で徹底的に解説します。これらの端子も一般的にはプルアップしておきます。

以上をまとめて、もっとも一般的な各信号端子の処理回路例を図 15 に示します。

(トランジスタ技術 1994 年 6 月号に加筆、修正)



## 割り込みの受け付けから割り込み処理の開始まで

### Z80の割り込みシステムの動作

末木 豊/野口智樹

#### 割り込み処理とは何か

##### ● 割り込みとは

人間の日常生活でも、何かをしているときに急に別の何かをしなければならず、そちらの処理をしてからもとの戻ることがあります。例えば、仕事をしているときに電話がかかると、話が終わったらまた仕事に戻ります。これが**割り込み**です。

ではマイコンにおける割り込みとは何なのでしょう。この章では割り込み処理の考え方で、Z80の割り込み動作について解説します。

##### ● マイコンにおける割り込み処理

まず図1(a)に外部からデータを読み込むいちばん

基本的なプログラムの例を示します。これはI/Oポート STATUS のデータがLレベルになったら、I/Oポート DATA からデータを読み込み、処理をしてから、ふたたび同じ動作を繰り返すものです。このように、準備ができていないかどうか状態を調べて、その状態が変化したときに対応する処理を実行するやりかたを**ポーリング**と呼びます。

しかしもしここで、STATUS が長い間Lレベルにならなかったら、CPU はその間じゅうループしていることになります。これではせっかくの **CPU の処理能力をむだにしている**ことになります。

いまここで処理すべき事柄が、このデータの読み込みしかないのであればこれでも問題はありますが、他にも処理すべき事柄があるときは、図1(b)のよう

```
LOOP: IN A, (STATUS)
      OR A
      JP NZ, LOOP
      IN A, (DATA)
```

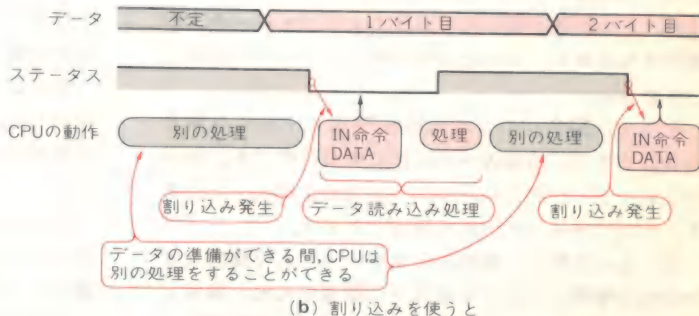
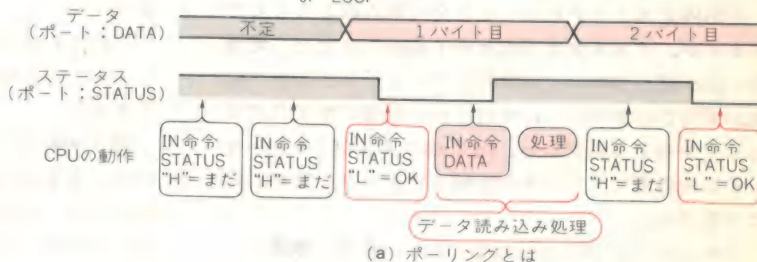
データの準備ができないとCPUはこの3行の命令を実行するだけ

処理

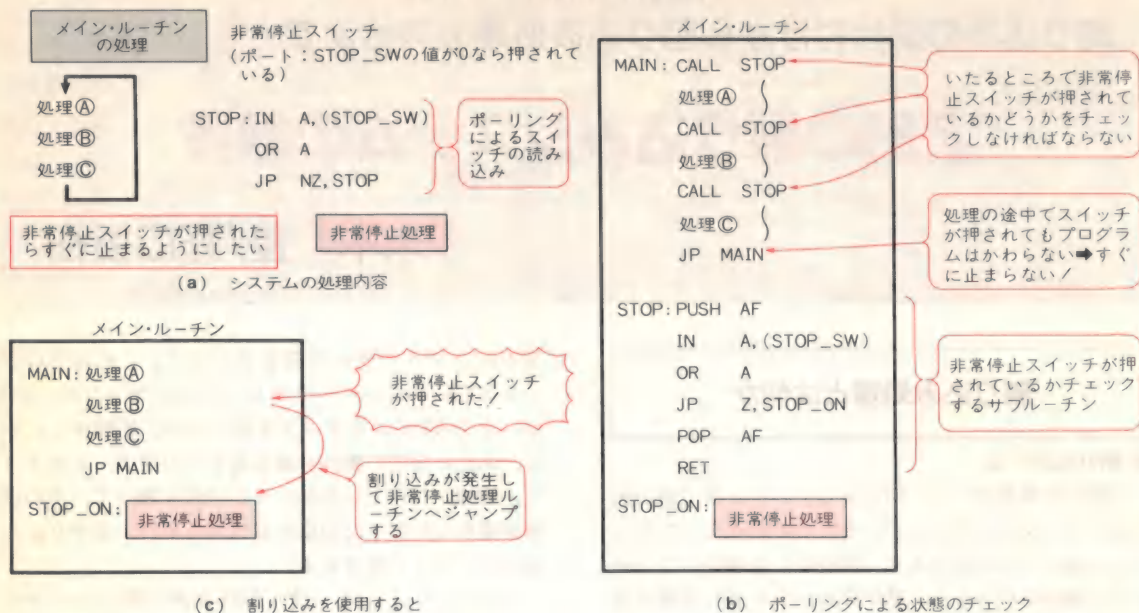
CPUの処理能力をむだに使っているノ

JP LOOP

図1  
ポーリングと割り込み



〈図2〉 ポーリングでは難しい処理



にするとどうでしょう。

これはステータス信号の立ち下がりをはきかけにしてCPUに信号を送り、その瞬間だけデータの読み込み処理をしてもらいます。処理が終わったらもとの処理に戻します。これが、**割り込み処理**の一例です。

### ● ポーリングでは難しい処理

次に図2をみてください。これは処理①～③を繰り返して実行しながら、非常停止するというマイコン・システムの例です。

非常停止スイッチはポーリングで読み取ることもできますが、チェックする箇所が1箇所だけだと、処理①～③が終わらないと非常停止スイッチのチェック処理が実行されませんし、それぞれの処理の始まりでチェックするようにすると、図2(b)のようにメイン・ルーチンの各所でスイッチの状態をチェックしなければなりません。

これでは**プログラムが大変**ですし、さらに厳密に言えば、各処理ルーチンの実行中にスイッチが押された場合、その処理が終わらないとチェック・ルーチンが実行されません。これでは**非常停止スイッチの意味がありません**。

そこで、図2(c)のように非常停止スイッチを割り込みにすると、処理①～③のどこを実行中でも、**非常停止スイッチが押された瞬間に非常停止処理を実行**することができるシステムを構築できます。

割り込み処理は、電源の異常やシステムの異常などの緊急な事態に対処する場合や、低速な入出力機器とのデータ転送を、待ち時間なしに効率よく行う場合、

またいつ発生するかわからない事柄を、別の処理をしながら待つときなどに使います。

このように割り込みとは、CPUのもつ処理能力をむだなく使用することのできる、マイコン・システムにおける重要な概念です。

### ● 割り込み処理のプログラム

すでに説明したように割り込み処理は、それまでの実行中のプログラムの流れを突然変え、用が済んだらふたたび何事もなかったかのように戻らなければなりません。

このためCPUの**レジスタやメモリを不用意に破壊しないように考慮**して、プログラムを組まなければなりません。

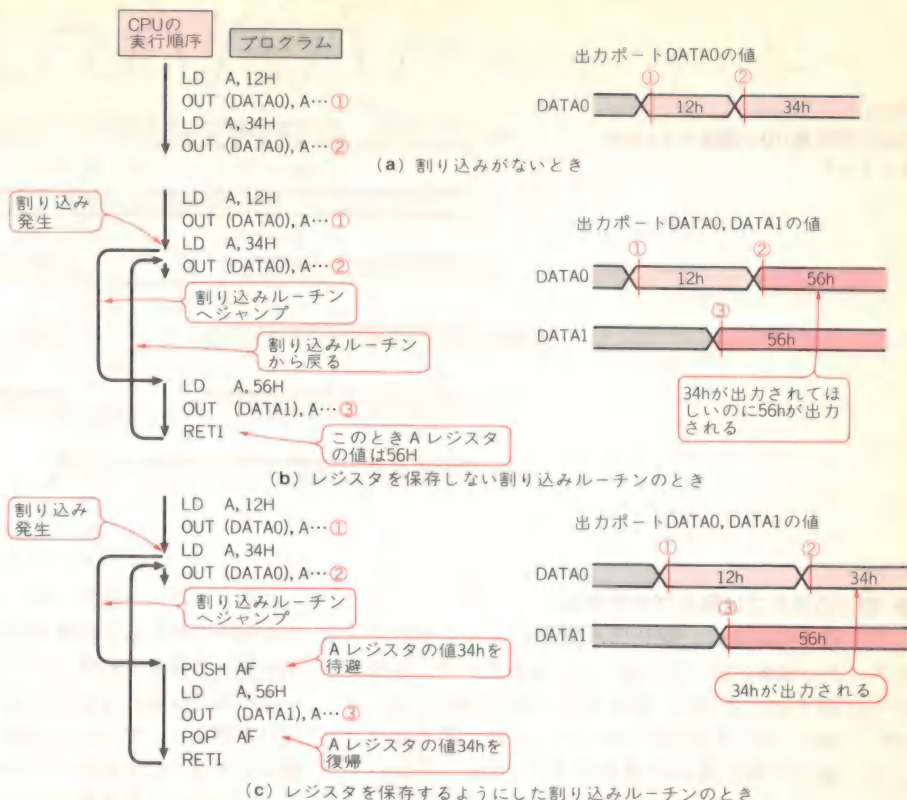
図3(a)を見てください。割り込みがないときは、I/OポートDATA0に12h→34hとデータが出力されるはずが、割り込みの発生するタイミングによっては、12h→56hとデータが出力されてしまいます。これはAレジスタに34hを代入したとたん割り込みルーチンにジャンプし、さらに割り込みルーチンでAレジスタを使っているためです。

このようなことのないよう、割り込みルーチンでは**使用するレジスタやフラグはすべて待避**して、**割り込み処理終了後に復帰してから戻るように**します [図3(c)]。

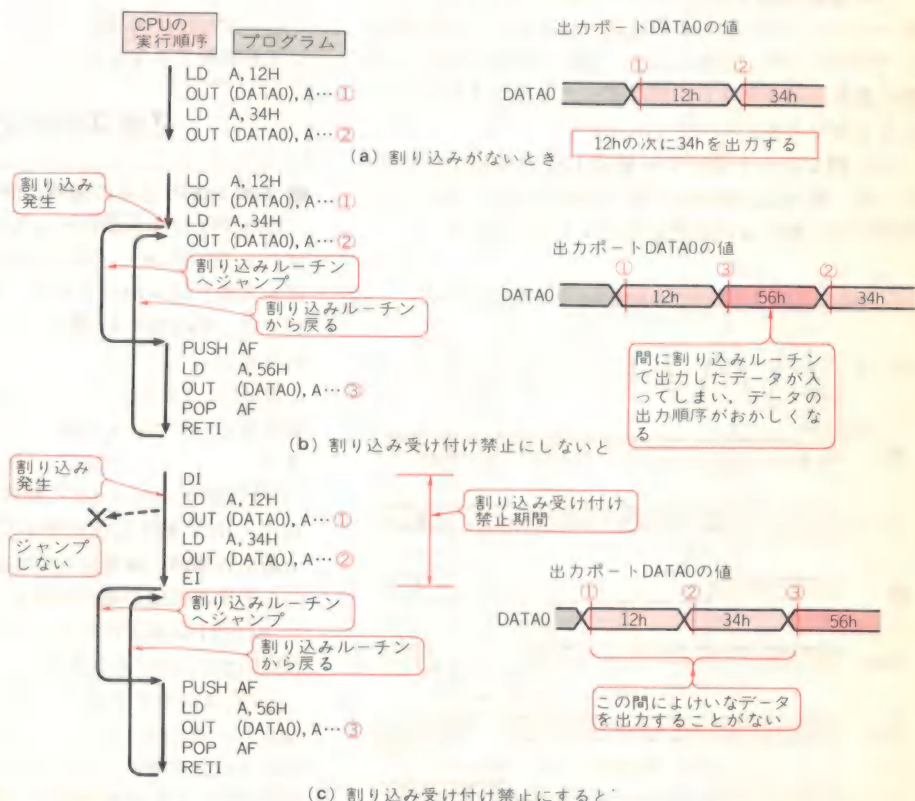
このように基本的には割り込み処理の先頭ですべてのレジスタをスタックに退避し、処理が終わったら退避していたレジスタを戻して実行途中だったプログラムに戻ります。



〈図3〉  
割り込みルーチンでレジスタを保存しておく

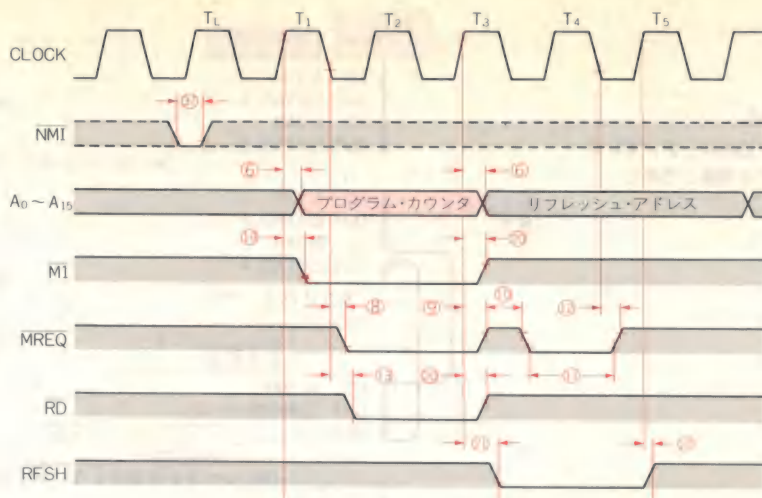


〈図4〉  
割り込まれたくないプログラムもある



〈図5〉

Z80のNMI割り込み認識サイクルの  
タイミング



### ● 割り込まれては困るプログラム

また図4のように、割り込み処理によって出力するデータの順番が狂ってしまうこともあります。本来ならば図4(a)のように、I/OポートDATA0には、12h→34hとデータを出力したいところが、図4(b)のように途中で割り込みが発生すると、12h→56h→34hと出力してしまうことがあります。

この場合は図4(c)のように、途中で割り込みルーチンにジャンプしないように**割り込み受け付けを禁止して対処**します。これにより、12h→34hを出力する間に割り込みが発生し、別なデータを出力されることがなくなります。

また割り込みを発生する周辺LSIの初期化プログラムも、割り込み受け付け禁止状態で実行しないと、意味のない割り込みが発生することがあります。

このように、一度何かをしだしたら、あるところまでは途中で割り込み処理などにジャンプしないで、一気に続けてほしい処理というものがあります。

このような場合はその処理に入る前に割り込み受け付けを禁止し、できるだけ短い時間で処理をすませて割り込み受け付けを許可します。

この“できるだけ**割り込み受け付け禁止状態を短くする**”というのがポイントで、不必要に長い時間、割り込み受け付けを禁止したままでは、割り込みが発生しても処理できなくなります。

## Z80の割り込み制御

### ● Z80の割り込みの種類と使い分け

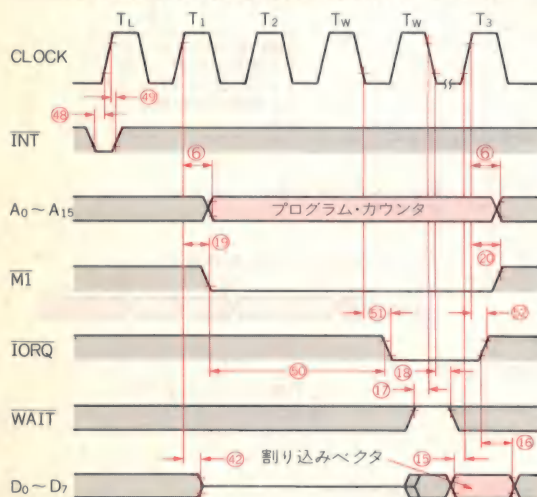
Z80CPUには**NMI**(Non Maskable Interrupt)と呼ばれる割り込みと、**INT**(Maskable Interrupt)と呼ばれる割り込みがあります。40ピンのDIPパッケージでは、それぞれ17番ピンと16番ピンに入力端子があります。

**NMI**はプログラムでマスク(割り込みの受け付けを禁止する)できないため、このような名前になっています。

**NMI**割り込みは**INT**割り込みよりも優先順位が高く、**INT**割り込み処理中でも受け付けられるので、一般には電源の異常に対応する処理などのような、もっとも優先順位の高い処理として使用されます。ただし**BUSREQ**端子がアクティブになっている場合には、**NMI**は受け付けられません。

**INT**はプログラムで、割り込み処理の受け付けを禁止したり許可したりすることが可能です。**INT**割り込みは、当然のことながらこれより優先順位の高い**NMI**割り込みの処理中や、**BUSREQ**端子がアクテ

〈図6〉Z80  $\overline{\text{INT}}$  割り込み認識サイクルのタイミング





〈表 1〉 Z80 の割り込み認識サイクル(単位: ns)

番号	記 号	項 目	6MHz 版		8MHz 版		10MHz 版		20MHz 版	
			最小	最大	最小	最大	最小	最大	最小	最大
6	$t_{dCr(A)}$	クロック立ち上がりからの有効アドレス出力遅延		90		80		65		57
8	$t_{dCr(MREQ)}$	クロック立ち下がりからの $\overline{MREQ} = "L"$ になるまでの遅延		70		60		55		40
9	$t_{dCr(MREQ)}$	クロック立ち上がりからの $\overline{MREQ} = "H"$ になるまでの遅延		70		60		55		40
10	$t_{wMREQH}$	$\overline{MREQ} = "H"$ のパルス幅	65		45		30		10	
11	$t_{wMREQ}$	$\overline{MREQ} = "L"$ のパルス幅	132		100		75		25	
12	$t_{dCr(MREQ)}$	クロック立ち下がりから $\overline{MREQ} = "H"$ になるまでの遅延		70		60		55		40
13	$t_{dCr(RD)}$	クロック立ち下がりから $\overline{RD} = "L"$ になるまでの遅延		80		70		65		40
15	$t_{dCr(CP)}$	クロック立ち上がりに対するデータ・セットアップ時間	30		30		25		12	
16	$t_{dCr(RD)}$	$\overline{RD}$ 立ち上がりに対するデータ・ホールド時間	0		0		0		0	
17	$t_{sWAIT(CD)}$	クロック立ち下がりに対する $\overline{WAIT}$ 信号セットアップ時間	60		50		20		7.5	
18	$t_{hWAIT(CD)}$	クロック立ち下がり後の $\overline{WAIT}$ ホールド時間	10		10		10		10	
19	$t_{dCr(MI)}$	クロック立ち上がりから $\overline{MI} = "L"$ になるまでの遅延		80		70		65		45
20	$t_{dCr(MI)}$	クロック立ち上がりから $\overline{MI} = "H"$ になるまでの遅延		80		70		65		45
21	$t_{dCr(RFSH)}$	クロック立ち上がりから $\overline{RFSH} = "L"$ になるまでの遅延		110		95		80		60
22	$t_{dCr(RFSH)}$	クロック立ち上がりから $\overline{RFSH} = "H"$ になるまでの遅延		100		85		80		60
37	$t_{wNMI}$	$\overline{NMI}$ パルス幅	60		60		60		60	
42	$t_{dCr(DZ)}$	クロック立ち上がりからデータ・バス・フロート状態までの遅延		80		70		65		40
48	$t_{sINT(Cr)}$	クロック立ち上がりに対する $\overline{INT}$ セットアップ時間	70		55		50		15	
49	$t_{sINT(Cr)}$	クロック立ち上がり後の $\overline{INT}$ ホールド時間	10		10		10		10	
50	$t_{dMI/(\overline{IORQ})}$	$\overline{IORQ}$ 立ち上がりに先き立つ $\overline{MI}$ 出力(L)の確定時間	359		270		220		100	
51	$t_{dCr(\overline{IORQ})}$	クロック立ち下がりから $\overline{IORQ} = "L"$ になるまでの遅延		70		60		55		45
52	$t_{dCr(\overline{IORQ})}$	クロック立ち下がりから $\overline{IORQ} = "H"$ になるまでの遅延		70		60		55		45

イブになっている場合には受け付けられません。

また **INT** 割り込みにはモード 0~2 の三つのモードがあります。モード 0 は Z80 の原型ともなった 8080A という CPU と互換性をもたせるためのモードです。またモード 1 も動作原理はモード 0 と同じです。

モード 2 が Z80 本来の割り込みモードで、このモード 2 と Z80 周辺 LSI を組み合わせることによって、他の CPU 系には見られない独特な(?)割り込みシステムを構成することができます。

図 5 と図 6、そして表 1 に、Z80 の割り込み認識サイクルを示します。

#### ● 割り込み制御に関係のある命令

Z80 の数ある命令のうち、割り込みの制御を行う命令が 10 個ほどあります。それぞれを簡単に説明すると、表 2 のようになります。

表中の P/V は、パリティ・フラグのことです。また  $IFF_1$  は、 $\overline{INT}$  割り込み処理の受け付け許可/禁止を制御する CPU 内部にあるフリップフロップで、 $IFF_2$  は、 $\overline{NMI}$  発生時に  $\overline{INT}$  割り込み受け付けが許可されているか禁止されているかを保存しておくフリップフロップです。

**IM 0~IM 2** は割り込みモードの設定です。またマスク可能な割り込みですから、割り込み受け付けを禁止する命令である **DI**、許可する命令である **EI** があります。

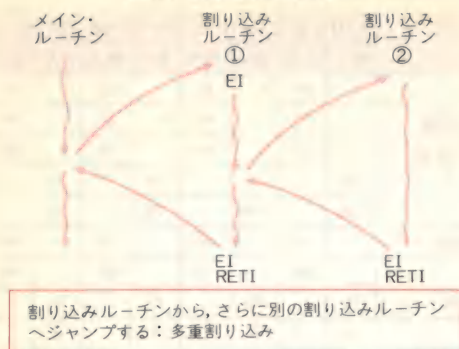
〈表 2〉 Z80 の割り込み制御に関係のある命令

命 令	動 作
<b>IM 0</b>	$\overline{INT}$ 割り込みをモード 0 に設定する。
<b>IM 1</b>	$\overline{INT}$ 割り込みをモード 1 に設定する。
<b>IM 2</b>	$\overline{INT}$ 割り込みをモード 2 に設定する。
<b>EI</b>	$\overline{INT}$ 割り込み受け付けを許可する。 $IFF_2 \leftarrow 1, IFF_1 \leftarrow 1$
<b>DI</b>	$\overline{INT}$ 割り込み受け付けを禁止する。 $IFF_2 \leftarrow 0, IFF_1 \leftarrow 0$
<b>RETI</b>	$\overline{INT}$ 割り込み処理からの復帰。
<b>RETN</b>	$\overline{NMI}$ 割り込み処理からの復帰。
<b>LD A, I</b>	I レジスタの内容を A レジスタに転送する。 $P/V \leftarrow IFF_2$
<b>LD I, A</b>	A レジスタの内容を I レジスタに転送する。
<b>LD A, R</b>	R レジスタの内容を A レジスタに転送する。 $P/V \leftarrow IFF_2$

また基本的に割り込み処理プログラムは、メイン・ルーチンとは別なプログラムで処理されます。つまりその割り込み処理ルーチンからメイン・ルーチンに戻るための、専用の **RET** 命令があります。 $\overline{NMI}$  処理ルーチンから戻るときに使われる **RETN**、 $\overline{INT}$  処理ルーチンから戻るときに使われる **RETI** です。

またモード 2 で使用されるインタラプト・レジスタへの値の設定は、**LD I, n** という直接値を代入する命令がありません。かならず A レジスタに設定してから I レジスタに代入します。

〈図7〉 多重割り込みとは



ここでリフレッシュ・レジスタである R レジスタの内容を A レジスタに読み込む LD A, R という命令が、なぜ割り込み制御に関係する命令なのでしょう。

その秘密は実行後のフラグの変化にあり、パリティ・フラグに IFF<sub>2</sub> の内容がコピーされるのです。つまり現在 Z80 が割り込み受け付け禁止状態であるか許可状態であるかを、プログラムでチェックできるので

また割り込みのモードは、Z80CPU 内部の IMF<sub>a</sub> と IMF<sub>b</sub> の二つのフリップフロップに格納されます。表 3 に IMF<sub>a</sub> と IMF<sub>b</sub> の割り込みモードとの関係を示します。リセット直後の状態は 8080A との互換性のために、モード 0 になります。

以上から、現在割り込み受け付け禁止状態であるか許可状態であるかは、LD A, R 命令などのフラグの変化で知ることができますが、現在の割り込みモードが何であるかをプログラムが知ることはできません。IMF の状態を読み出す命令がないからです。

### ● 多重割り込み

Z80 には  $\overline{\text{NMI}}$  と  $\overline{\text{INT}}$  の 2 本の割り込み入力端子がありますが、 $\overline{\text{NMI}}$  は緊急事態通知用として考えると、通常使える割り込みは一つしかありません。

となると優先度の高い割り込みも優先度の低い割り込みも、同一の割り込み入力端子に接続されることになります。

もし優先度の低い割り込み処理が先に発生し、その処理が終わるまで優先度の高い割り込みが待たされてしまうのでは、何のための優先順位かわかりません。優先度の低い割り込み処理中には、優先度の高い割り込みを許可しておく必要があります。

このように割り込み処理中に、別の割り込み処理を行うことを、**多割り込み処理**といいます(図 7)。

実際には、割り込み処理ルーチンの先頭付近で、EI 命令を実行することで割り込み受け付け許可状態にします。しかし単純に割り込み受け付けを許可してしまうと、優先度の低い割り込みまで受け付けてしまいそ

〈表 3〉 IMF<sub>a</sub>、IMF<sub>b</sub> と割り込みモードの関係

IMF <sub>a</sub>	IMF <sub>b</sub>	割り込みモード
0	0	モード 0
0	1	未使用
1	0	モード 1
1	1	モード 2

うです。

このためモード 0 やモード 1 の割り込み処理の場合は、プログラムの優先順位を管理する必要があります。

Z80 本来の割り込みモードであるモード 2 と Z80 周辺 LSI の組み合わせでは、この優先順位をハード的に決定できる(デジィ・チェーン接続)ので、多重割り込み処理の管理が非常に簡単になります(多重割り込みやデジィ・チェーンについては後述)。

## NMI 割り込み

### ● $\overline{\text{NMI}}$ 割り込みとは

Z80 の割り込み制御のうち比較的動作が簡単で理解しやすいのが  $\overline{\text{NMI}}$  割り込みです。

すでに述べたように、本来  $\overline{\text{NMI}}$  は電源の異常やメモリのパリティ・エラーが発生した場合などの、緊急事態を CPU に知らせるための割り込みとして使われます。

しかし、これは Z80 をコンピュータの CPU として使っていた頃の考え方で、別に必ず電源異常のチェックに使わなければならないという決まりはどこにもありません。割り込みが 2 本あれば間に合うシステムなら、一般的な割り込み処理として  $\overline{\text{NMI}}$  を使ってもかまわないでしょう。

しかし産業用機器のコントローラとして使う場合や、信頼性を少しでも向上させたいときには、 $\overline{\text{NMI}}$  をこれらの目的に使うのが正統的な使い方です。

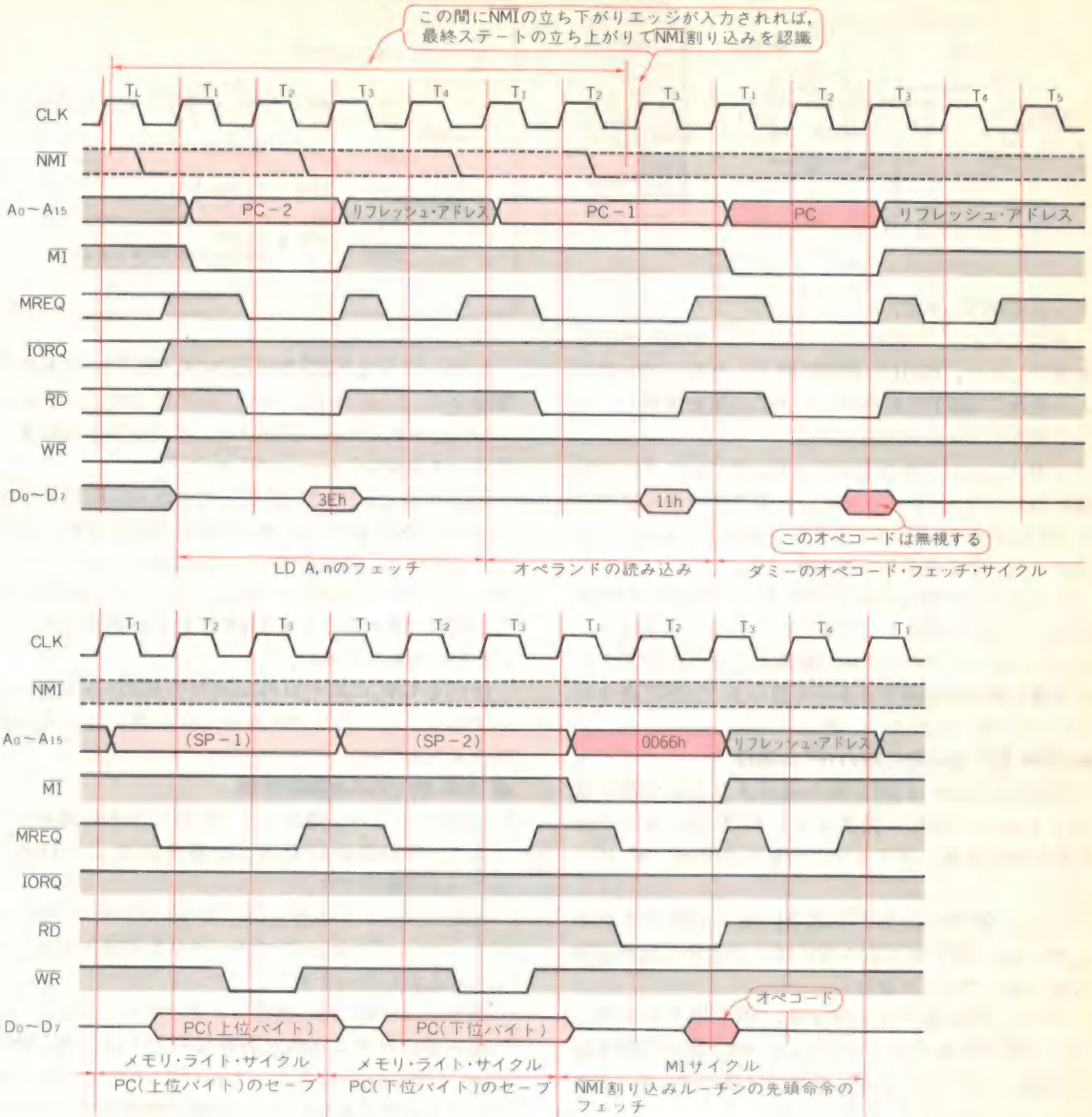
$\overline{\text{NMI}}$  は優先度が高いといっても、**BUSREQ 端子がアクティブになっている場合は受け付けられません**。BUSREQ 端子がアクティブな状態とは、CPU 以外の別のコントローラが、CPU が制御しているバスの制御権を要求しているということで、例えば DMA (Direct Memory Access) コントローラなどが、メモリの読み書きをしたいことを CPU に通知している状態です。

BUSREQ がアクティブになったら、CPU はそのときの命令を実行後、すみやかにバスの制御権をゆずらなくてはなりません。

よって CPU は、バスの制御権を渡した後は  $\overline{\text{NMI}}$  だろうがなんだろうが、制御権が返されるまでは動くことすらできません。



〈図8〉  $\overline{\text{NMI}}$  割り込みのバスの動き



またある意味で、あらゆるCPUにとって最上位の割り込みはリセット入力であるとも考えられます。Z80の場合、どんなバス・サイクルからでも割り込みがかけられ、割り込みモードは0に、割り込み受け付けは禁止に、そして有無をいわずアドレス0000hに実行が移されるわけですから…。

#### ● $\overline{\text{NMI}}$ 割り込みのバスの動作

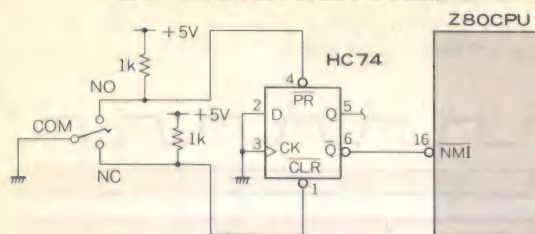
$\overline{\text{NMI}}$  入力信号はエッジ・トリガで、立ち下がりエッジが基準となります(エッジ・トリガ)。立ち下がりエッジの後のLレベルの期間は、CPUのクロック・スピードにより異なります。例えば4MHz動作のZ80で最低80ns以上なので、外部からの信号としては80ns以上のLレベル

のパルスを入力すれば、 $\overline{\text{NMI}}$  割り込み受け付けのCPU内部のフラグをONすることができます。

図8に $\overline{\text{NMI}}$  割り込みのバスの動作を示します。このフラグは、直前の命令のマシン・サイクルの最終ステートの立ち上がりでチェックされ、 $\overline{\text{NMI}}$  割り込みが認識されます。このタイミングは $\overline{\text{INT}}$  割り込みも共通です。

そして、次に実行しようとしていたプログラム・カウンタのアドレスの命令を読み込みます。しかしここで読み込んだオペコードは、CPUには取り込まれず破棄されます。そしてPCの上位バイト、PCの下位バイトをスタックに待避すべくメモリ・ライト・サイク

〈図9〉  $\overline{\text{NMI}}$  割り込みを発生させる回路



ルを2回実行します。

そしてアドレス 0066h にジャンプし、0066h の命令を実行します。 $\overline{\text{NMI}}$  の処理が終了したら、プログラムの最後に RETN を実行して PC の値を復帰し、もとの実行アドレスに戻ります。

このタイミングをみるとわかるように、スタックを積む前にダミー・サイクルとしてオペコードを読み込んでいるのがわかります。またこのダミー・サイクルが5クロックである点も不思議です。

これはここで読み込んだ命令を、内部的には RST 66H(?) という命令として実行していると考えると、この5クロック・サイクルの意味と、その次のスタックを積む動作が理解できるかと思います(RST 命令のマシン・サイクルは5, 3, 3)。

### ● $\overline{\text{NMI}}$ 割り込み発生時の IFF の動作

$\overline{\text{NMI}}$  割り込みは  $\overline{\text{INT}}$  割り込みより上位の割り込みですから、 $\overline{\text{NMI}}$  が発生すると  $\overline{\text{INT}}$  割り込みは受け付け禁止状態になります。つまり  $\text{IFF}_1$  が“0”になります。

しかし RETN 命令で  $\overline{\text{NMI}}$  割り込み処理から戻るときには、 $\overline{\text{INT}}$  割り込み受け付けの状態も元の状態に戻さなくてはなりません。

つまり  $\overline{\text{NMI}}$  割り込み発生前、 $\overline{\text{INT}}$  割り込み受け付けが許可状態であったならば、RETN 実行後も許可状態にもどさなければ、受け付け禁止の状態のままになってしまいます。

このために  $\text{IFF}_1$  と  $\text{IFF}_2$  の二つのフラグがあり、RETN 命令は  $\text{IFF}_2$  の状態を  $\text{IFF}_1$  にコピーして  $\overline{\text{INT}}$  割り込みの受け付け状態を戻します。

この動作のおかげで、 $\overline{\text{NMI}}$  処理の後でも  $\overline{\text{INT}}$  割り込みを受け付けることができます。よって  $\overline{\text{NMI}}$  割り込み処理終了は必ず RETN 命令を実行するようにします。

### ● RETN 命令と $\overline{\text{NMI}}$ 割り込み

もし  $\overline{\text{NMI}}$  処理プログラムが長く、 $\overline{\text{NMI}}$  処理中に再び  $\overline{\text{NMI}}$  割り込みが発生したらどうなるでしょうか。 $\overline{\text{NMI}}$  処理プログラムの最後には RETN 命令を入れますが、これが実行されるまでは  $\overline{\text{NMI}}$  は再度発生しないようにも思えます。

〈リスト1〉  $\overline{\text{NMI}}$  割り込みで LED を反転表示するプログラム

```

org      0000h

START:   ld      sp, 0      ; スタート
         ld      a, 0      ; スタック設定
LABEL1:  jp      LABEL1    ; 無限ループ

NMITOP:  org      0066h    ;  $\overline{\text{NMI}}$  割り込み
         xor     out      01h      ; LED 点灯
         out     (20H), a      ; LED 点灯
         retn                ;  $\overline{\text{NMI}}$  割り込みルーチンから戻る
    
```

しかしじつは  $\overline{\text{NMI}}$  割り込みは、RETN の実行に関係なく、 $\overline{\text{NMI}}$  割り込み入力端子に立ち下がりエッジが入力されるたびに認識され、その命令を実行後にアドレス 0066h にジャンプします。

RETN 命令はすでに説明したように、 $\text{IFF}_2$  に保存してある  $\overline{\text{INT}}$  割り込み受け付けの状態を  $\text{IFF}_1$  に戻すという処理と、通常の RET 命令と同じスタックから戻りアドレスを POP して戻ることをしているだけで、これが実行されるまで  $\overline{\text{NMI}}$  割り込みが再度発生しないわけではありません。

$\text{IFF}_2$  を  $\text{IFF}_1$  に戻す以外は通常の RET 命令となんら変わりなく、また Z80 周辺 LSI に対しても特別な意味をもちません。

### ● $\overline{\text{NMI}}$ 割り込み制御回路例

$\overline{\text{NMI}}$  割り込みを発生させる簡単な回路を図9に示します。回路図の CPU のピン番号は 40 ピン DIP パッケージの番号です。

手動のスイッチの接点出力を SR-FF(セット/リセット・フリップフロップ)でチャタリング除去処理したものを入力しています。

厳密にはパルス幅は確定できていませんので、パルス幅を確定させる回路が必要なのですが、世の中に 60 ns 未満の時間でスイッチを ON/OFF できる人間がいるとは思えませんので、回路の簡略化のためにここでは省略します。

また割り込み信号を入れて割り込みが認識されても、それは CPU 内部の話なので動作の確認ができません。そこで図10に示すような、簡単な出力回路を作り、 $\overline{\text{NMI}}$  の動作を確認してみます。

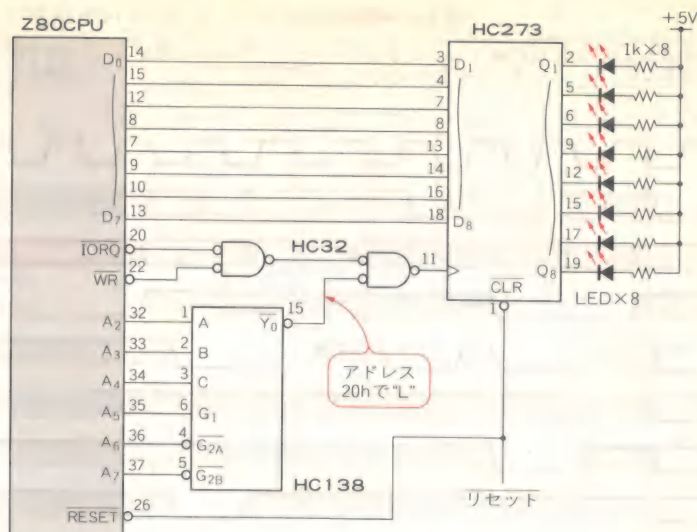
図10の I/O 回路については第4章で解説済みなのでここでは触れません。I/O アドレスは 20h です(正確にはイメージ・アドレスが発生しているのが 20h ~ 23h となる)。

### ● $\overline{\text{NMI}}$ 割り込み処理プログラム

リスト1に  $\overline{\text{NMI}}$  割り込みで LED を反転表示するプログラムを示します(メモリについては ROM32 K バイト、RAM32 K バイトの Z80 システムを想定して



〈図 10〉  
LED 点灯回路



いる)。INT 割り込みがどのモードであろうとも NMI が上位の割り込みであることにかわりありません。

プログラムの動作を説明します。まずリセット直後は、アドレス 0000h から実行が開始されます。まずスタック・ポインタをセットします。そして表示データを保持する A レジスタに 0 を代入して値をクリアしています。

そして次のアドレス 0006h には、0006h にジャンプする命令を書いています。つまりプログラムの流れとしては、アドレス 0006h へ永久にジャンプを繰り返すだけで何もしていません。

さて、ここで外部からの NMI 割り込みが発生すると、プログラム実行アドレスはハードウェア的に分岐され、アドレス 0066h に実行が移ります。

A レジスタと 01h との排他的論理和(XOR)を計算しているので、A レジスタの最下位ビットだけが反転します。この値を I/O アドレス 20h に OUT 命令で出力します。そして RETN 命令で復帰します。

このプログラムを実行すると、スイッチが押されるたびに最下位ビットの LED がついたり消えたりします。

この例は割り込みの動作チェックのためのものなので、割り込みルーチンの先頭でレジスタの値を保存していません。しかし、実際の割り込み処理では必ずレジスタの値は保存してください。

## INT 割り込み

### ● INT 割り込みとは

NMI 割り込みと違い、INT 割り込みは通常の割り

込み処理用の入力端子です。またすでに述べたように Z80 にはモード 0～2 の割り込みモードがあります。

Z80 のリセット直後は、INT 信号に対する割り込み受け付けが禁止状態となっています。また割り込みのモードはモード 0 です。

モード 1 またはモード 2 の割り込みを使用する場合には、IM1 または IM2 命令を実行する必要があります。

### ● INT 割り込みのしくみ

NMI 割り込みと違うところは、INT 入力信号の L レベルが基準となることです(レベル・トリガ)。

CPU は命令の最後のステートの立ち上がりで INT 入力をサンプリングし、L レベルかどうかをチェックします。

ここで INT 割り込みが認識されれば、次に割り込みモードによって、各モードの割り込みの応答サイクルが始まります。

また INT 割り込みは、プログラムによって受け付けを禁止したり許可したりできます。

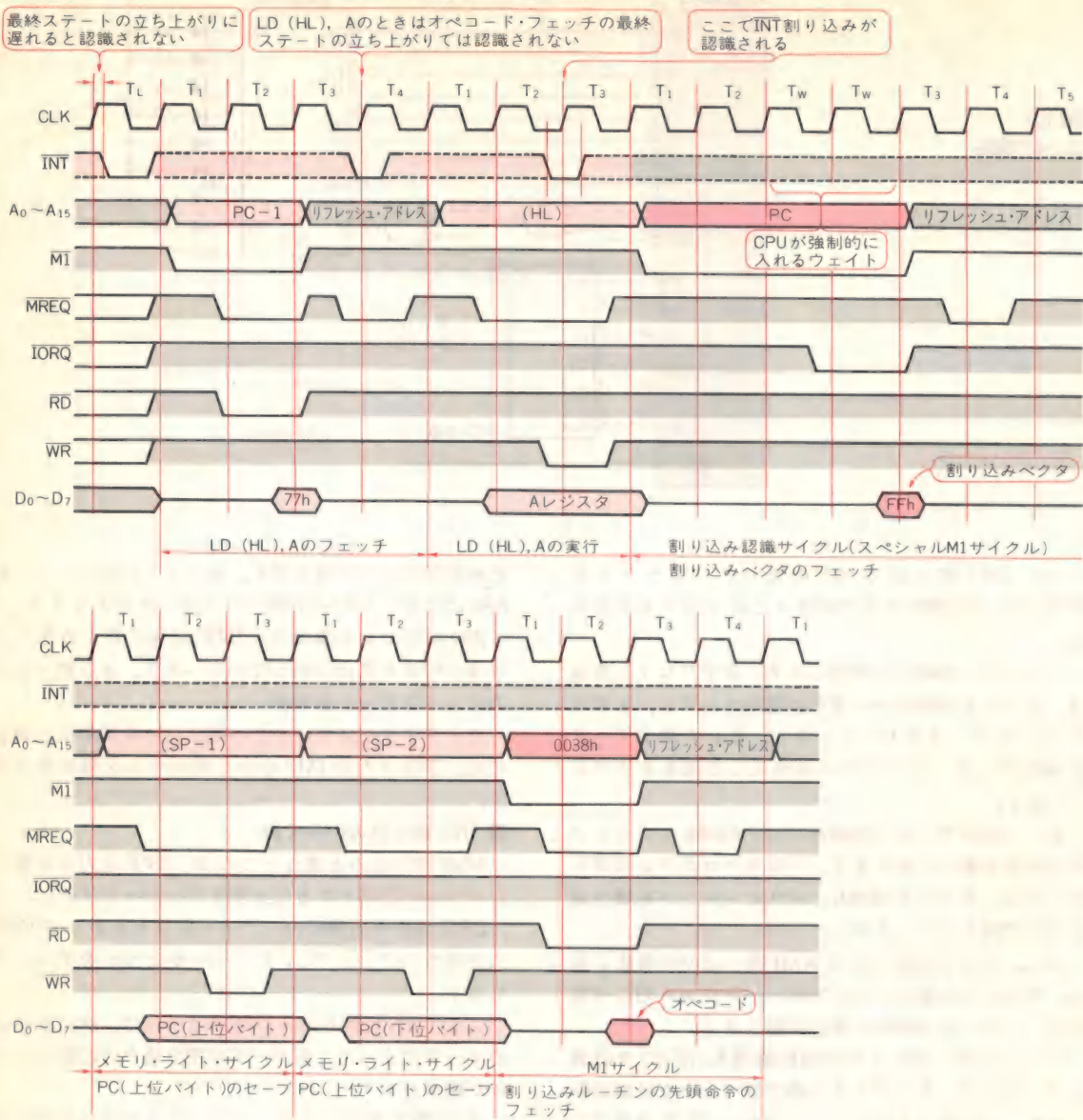
割り込み受け付けを許可するには EI 命令を実行します。禁止するには DI 命令を実行します。またリセット信号の入力や NMI の受け付けによっても禁止できます。

そして重要なのは、INT 割り込み自身が発生したときにも、再度 INT 割り込みが発生しないように受け付けが禁止されます。つまり一度 INT が発生すると、EI 命令を実行しない限りもう一度 INT 割り込み信号が入力されても認識されません。これも NMI と違う点です。

表 4 に IFF<sub>1</sub> と IFF<sub>2</sub> の動作をまとめます。

それではモード 0～2 をそれぞれ説明します。

〈図 11〉  $\overline{\text{INT}}$  割り込み (モード 0, モード 1) のバス動作



## モード0 割り込みの動作

### ● モード 0 割り込みとは

モード0割り込みは8080Aと同じ動作をする割り込みです。といっても1973年に発表され、Z80や8085の出現で急速に製品寿命を閉じた8080CPUを、どれだけの人が知っているか(名前だけでなくその動作まで)はわかりませんが…。CPUチップの実物はおろか、データ・シートさえ見たこともない人がほとんどでしょう。

### ● モード 0 割り込みのバスの動作

割り込を受け付けるタイミングは  $\overline{NMI}$  と同一ですが、割り込み応答サイクルに特徴があります。

図 11 にモード 0 のときの、 $\overline{\text{INT}}$  割り込みのバスの動作を示します。

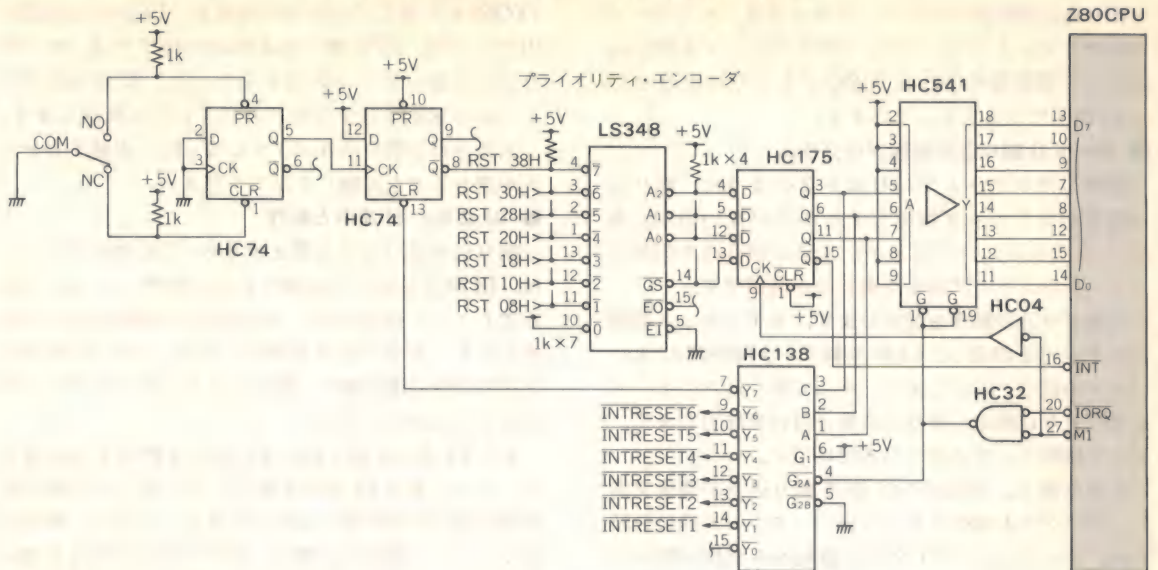
$\overline{\text{INT}}$  割り込み認識後、CPU は  $\overline{\text{MI}}$  と  $\overline{\text{IORQ}}$  を L レベルにして、データ・バスから割り込みベクタを読み込みます。

モード0のときの割り込みベクタとは、Z80が実行できる命令(オペコード)です。

この割り込みベクタとして使う命令は、Z80 の命令



〈図 12〉  $\overline{\text{INT}}$  割り込み(モード 0)を発生させる回路



〈表 4〉 IFF<sub>1</sub>と IFF<sub>2</sub>の動作

	IFF <sub>1</sub>	IFF <sub>2</sub>
リセット	0	0
DI 命令	0	0
EI 命令	1	1
$\overline{\text{INT}}$ 割り込み処理	0	0
$\overline{\text{NMI}}$ 割り込み処理	0	変化なし
RETN 命令	IFF <sub>2</sub> の状態がコピーされる	変化なし
RETI 命令	変化なし	変化なし

0:  $\overline{\text{INT}}$  割り込み受け付け禁止,  
1:  $\overline{\text{INT}}$  割り込み受け付け許可

であれば基本的にはどんな命令でもよいのですが、プログラムの分岐およびプログラム・カウンタのスタックへの待避を考えると、**RST 命令**か **CALL 命令**が適します。

RST 命令は 8 種類あり、それぞれアドレス 0000h, 0008h, 0010h, 0018h, 0020h, 0028h, 0030h, 0038h をコールしプログラムを実行します。RST 命令は 1 バイト命令なので、データ・バス上に命令を出力する回路が簡単に実現できます。 $\overline{\text{INT}}$  割り込みの種類が 8 種類以下の場合便利です。

CALL 命令は 3 バイト命令なので、回路が複雑になるのでここでは取り上げません。

#### ● モード 0 では RST 命令が最適

図 11 の例に示すように、RST 命令を割り込みベクタとして使った場合は、CPU はこれを読み込み実行します。RST 命令は M1 サイクルを 5 クロックで実行し、また割り込みベクタ読み込み時は、CPU が

強制的に 2 クロックのウェイトを入れるので、合計 7 クロックの時間がかかります。

この割り込みベクタを読み込むときの動作は、 $\overline{\text{MREQ}}$  に替わって  $\overline{\text{IORQ}}$  がアクティブになり、それ以外のタイミングは通常のオペコード・フェッチと同等であることから、**スペシャル M1 サイクル**とも呼ばれています。

スペシャル M1 サイクル後は、通常の RST 命令の実行と同様、プログラム・カウンタをスタックに退避するためにメモリ・ライト・サイクルが 2 回実行されます。

そして RST 命令の飛び先である各アドレスからオペコード・フェッチを開始し、割り込み処理ルーチンの実行に移ります。

#### ● モード 0 割り込み制御回路例

モード 0 の  $\overline{\text{INT}}$  割り込みを発生させる外部回路を図 12 に示します。CPU のピン番号は 40 ピン DIP タイプの番号を使用しています。

HC タイプの汎用ロジック IC を数個使用して、8 種類の割り込み入力の優先順位付けと、RST 命令の発生を行っています。

RST 命令は 8 種類あるので、割り込みも 8 種類に対応できるのですが、**RST 00H は、リセット後と同様のアドレス 0000h からの実行なので通常はあまり使用しません。**ここでも RST 00H は使用していません。

動作確認のための LED 点灯回路は図 10 の回路をそのまま使います。

スイッチ回路出力 (HC74 の 8 番ピン) を RST 08H ~ RST 38H のどれかに接続してスイッチを押すと、対応した割り込みが発生します。

プライオリティ・エンコーダの入力に接続されている割り込み保持用のフリップフロップは、エンコードされた3ビットのコードと、CPUが出力する割り込みベクタ要求信号(MIとIORQ)がLレベルになったとき信号でリセットしています。

### ● モード0割り込み処理プログラム

実際のプログラムではリスト2のように、割り込み処理が2ステップ程度ですむわけがないので、各RST命令のジャンプ先アドレスには割り込み処理ルーチンへのジャンプ命令を書くのが普通です。

プログラムの動作を説明します。まずリセット直後にスタックを設定し、LED点滅データ保持用のAレジスタの値をクリアします。そして割り込みのモードをモード0に設定し割り込み受け付けを許可します。そして無限ループさせているだけです。

この状態で、外部からのINT割り込みが発生すると、プログラム実行アドレスはハードウェア的に分岐され、それぞれのRST命令の番地から実行が開始されます。

それぞれの割り込み処理ルーチンでは、反転表示したいビットに相当するBレジスタのビットを1にして、LED点灯出力ルーチンへジャンプします。

〈リスト2〉INT割り込み(モード0)で七つのLEDを反転表示するプログラム

START:	org	0000h	
	jp	JMP	:スタート
RST08H:			
	org	0010h	:RST 08H
	ld	b, 02h	
	jp	INTRET	
RST10H:			
	org	0010h	:RST 10H
	ld	b, 02h	
	jp	INTRET	
RST18H:			
	org	0018h	:RST 18H
	ld	b, 04h	
	jp	INTRET	
RST20H:			
	org	0020h	:RST 20H
	ld	b, 08h	
	jp	INTRET	
RST28H:			
	org	0028h	:RST 28H
	ld	b, 10h	
	jp	INTRET	
RST30H:			
	org	0030h	:RST 30H
	ld	b, 20h	
	jp	INTRET	
RST38H:			
	org	0038h	:RST 38H
	ld	b, 40h	
	jp	INTRET	
INTRET:			
	xor	B	
	out	(20h), A	:LED点灯
	ei		:割り込み受け付け許可
	reti		:メイン・ルーチンへ戻る
JMP:			
	org	080H	
	ld	sp, 0	:スタック設定
	ld	a, 0	
	im	0	:割り込みモード0
	ei		:割り込み受け付け許可
LABEL1:			
	jp	LABEL1	:無限ループ

そしてAレジスタとBレジスタの排他的論理和(XOR)を計算してからその結果を、LED点灯回路に出力します。INT割り込み処理の終了では、次の割り込みの受け付けを許可するために、EI命令を実行してからRETI命令でもとのアドレスに復帰します。

入力される割り込みベクタによって、点滅するビットが違うことを実験してみてください。

### ● DI命令とEI命令と動作

割り込み受け付けを禁止するのがDI命令です。では、DI命令を実行中に割り込みが発生したらどうなるでしょう。DI命令は、命令実行中は割り込みが発生してもこの受け付けを禁止します。つまりDI命令と次の命令以後の間で、割り込みルーチンにジャンプすることはありません。

またEI命令は割り込み受け付けを許可する命令です。しかし実はEI命令を実行してもすぐには割り込み受け付け許可状態にはなりません。これは、割り込みルーチンの最後に必要な、RETI命令の実行を考え、EI命令の次の命令の実行後から受け付け許可状態になります。

ここで連続的に割り込みが発生したときを考えてみます。まず割り込みが発生するとメイン・ルーチンへ

**Q** : Z80の割り込み制御で、とくにモード2は非常に特徴的なシステム構成だと思う。このような割り込みシステムに設計した理由は？ 8080からの拡張ならば、RST命令を増やすなどの方法でもよかったのではないか。

**A** : Z80CPUは、ファミリ周辺デバイスも含めたトータル・システムを目指して設計されました。それ故、ファミリ周辺デバイスからの割り込みを効率的に処理できるように、モード2割り込みが付け加えられたのだと思います。割り込みコントローラを使ったシステムの場合、割り込みルーチンのプログラムが作りにくいというのもあるでしょう。

また8080の1バイト命令の未使用命令は、もうほとんど残っていないなかったので、さらに増やすとなると2バイト以上のRST命令になってしまいます。

以上から、外部から1バイトのベクタを読み込み、間接的にジャンプするこの方式が、周辺デバイスに付加する回路規模からも現実的だったのだと思います。



の戻りアドレスをスタックに PUSH し、割り込みルーチンに実行が移ります。そしてルーチンが終わったらスタックから、次の割り込み処理ができるように EI 命令を実行し、元のメイン・ルーチンの続きを実行するため RETI 命令を実行します。

このとき、EI 命令を実行した瞬間から割り込み受け付けが許可状態になると、次の RETI 命令を実行しないうちに再度割り込みルーチンの先頭にジャンプしてしまいます。これをそのまま繰り返すと、RAM がスタックでいっぱいになってしまいます。

では割り込みルーチンからメイン・ルーチンに戻る RETI 命令に、EI 命令と同等な動作を入れれば問題ないように思えます。

しかし RETI 命令に、割り込み受け付けを許可する動作を加えると、後述する多重割り込みの処理で不都合が出る場合があります。一般的な割り込み処理ルーチンの最後では、面倒でも EI 命令と RETI 命令を入れるようにします。

また EI 命令の次の命令の実行後から割り込み受け付けが許可されますから、**EI 命令と DI 命令が続いた場合は、割り込み受け付けは禁止のままとなります。**EI と DI の間に NOP 命令が 1 個でもあれば、そこで

割り込みを受け付けることができます。

この EI 命令と DI 命令の動作は、モード 1 でも 2 でも同じです。

## モード 1 割り込みの動作

### ● モード 1 割り込みとは

モード 1 割り込みの考え方は  $\overline{\text{NMI}}$  と同じで、割り込み信号が入力されたら特定のアドレスへジャンプするモードです。もし Z80 を使用して組み上げるシステムの割り込み処理が一つでよい場合は、モード 1 を使用するのも得策です。

また 2 種類使用する場合でも、片方は  $\overline{\text{NMI}}$  割り込みを使用するのがわかりやすくよいでしょう。

### ● モード 1 割り込みのバスの動き

モード 1 の  $\overline{\text{INT}}$  受け付け後の応答サイクルは、タイミング的には図 11 のモード 0 の例とまったく同じです。ただし、**読み取った割り込みベクタつまりオペコードは、どんなデータであろうとすべて無視し、RST 38H(オペコード：FFh) という命令に置き換えて実行されます。**つまりモード 1 はマスク可能な  $\overline{\text{NMI}}$  のアドレス 0038h 版とも考えられます。

## Z80 七不思議 割り込み編

**Q**：周辺 LSI が RETI 命令を認識する必要はないのではないか。またこのことによって、CPU と周辺 LSI の間にバッファを入れる場合など、回路が面倒になってしまう。

**A**：Z80 ファミリーは、周辺も含めて一つのシステムを構成するという考えに立って設計されたのだと思います。また、割り込み優先順位に関しても、その優先順位をダイナミックに変えなければならぬアプリケーションはほとんどないと考えて、割り込み優先順位はデジィ・チェーンで固定し、Z80 周辺 LSI は RETI 命令を受けて周辺デバイス間で優先順位を制御するという方式をとっています。

そのため、複数のボードにまたがった場合のハードウェアが面倒になるというデメリットより、周辺デバイスに割り込みの終了を認識させることにより処理の簡素化を図り、ソフトウェアのオーバ・ヘッドを減らすほうを取ったのだと思います。

**Q**：モード 2 の割り込みベクタが偶数アドレスでなければならないのはなぜか。他の CPU では割り込み

本数を稼ぐため、シフトしてから使用するものなどがある。

**A**：これも、できるだけゲート数を減らすためではないかと思います。

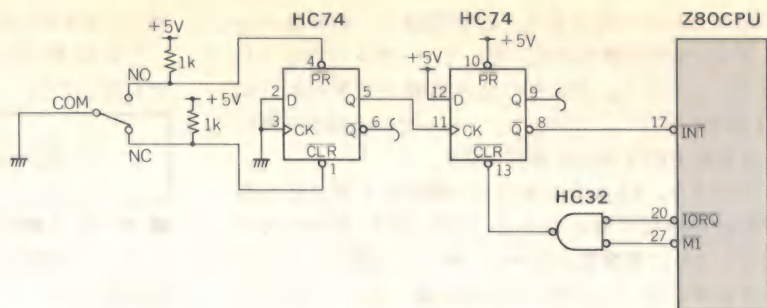
また Z80 設計当時、128 本以上の割り込みベクタが必要になるアプリケーションはほとんど存在しなかったのではないかと思います。ベクタが格納されているアドレスをシフト計算なしですぐに知ることができる、というメリットもあります。

**Q**：割り込みベクタ要求時に、なぜ  $\overline{\text{IORQ}}$  がアクティブになるのか、割り込みベクタ・リクエストなどのような専用信号線を入れたほうがわかりやすいし、回路が簡単になったのではないか。

**A**：これも信号線の節約というのが理由だと思います。Z80PIO のリセット入力や Z80SIO のボンディング・オプションなどをみるとわかるように、当時はパッケージの自由度が少なく、40 ピンなら必ず 40 ピンに収めなければならなかったためだと思われます。

〈信垣育司〉

〈図 13〉  
INT 割り込み(モード 1)を発生させる回路



そこでモード 1 で使用する場合は、基本的にはデータ・バスに割り込みベクタを出力しなくても、**INT 入力に L レベルの信号を入力するだけでアドレス 0038h へジャンプさせることができます。**

バスの動作としてはモード 0 と同じタイミングで動いています。スペシャル M1 サイクルを実行後、プログラム・カウンタの上位、下位をスタックに待避し、アドレス 0038h にジャンプします。

割り込み応答サイクルに挿入されている 2 クロックのウェイトや、 $\overline{\text{IORQ}}$  と  $\overline{\text{M1}}$  による割り込みベクタ要求信号は、本来モード 1 割り込みではバスからは何の情報も読み込まないので必要ありません。しかしモード 0、モード 2 とバス・サイクルを統一している(ハードウェアが共通になり、回路規模を小さくすることができる)ためだと思われます。

#### ● モード 1 割り込み制御回路例

モード 1 の  $\overline{\text{INT}}$  割り込みを発生させる簡単な外部回路を図 13 に示します。CPU のピン番号は 40 ピン DIP タイプの番号を使用しています。

基本的な考え方は図 9 の回路とあまり変わりありませんが、手動のスイッチの接点出力を SR-FF(セット/リセット・フリップフロップ)でチャタリング除去処理したものを、割り込み保持用のフリップフロップの入力としています。

この割り込み保持用のフリップフロップは、スイッチの押し下げでセットされ、CPU の割り込みベクタ要求信号( $\overline{\text{M1}}$  と  $\overline{\text{IORQ}}$  が同時に L レベルになる)でリセットしています。

また動作確認のための LED 点灯回路は、これも図 10 の回路を使います。

#### ● モード 1 割り込みの処理プログラム

いままでの例と同様に、モード 1 割り込みで LED を反転表示するプログラムを、リスト 3 に示します。

プログラムの動作を説明します。まずリセット直後にスタックの設定と表示データを保持する A レジスタをクリアします。そして割り込みモードをモード 1 に設定し割り込み受け付けを許可します。

そして無限ループを繰り返すだけで、他には何もしていません。

外部からの  $\overline{\text{INT}}$  割り込みが発生すると、プログラム実行アドレスはハードウェア的に分岐され、アドレス 0038h から実行を始めます。

A レジスタと 02h との排他的論理和を計算し、ビット 1 を反転させたあとに LED 点灯回路に出力します。またモード 0 と同様、次の割り込み受け付け許可のために EI 命令を実行し、RETI 命令で元のプログラムに戻ります。

#### ● モード 1 での多重割り込み

モード 0 では割り込みベクタとして RST 命令を使うことで、8 本の割り込みを制御することが可能でした。多重割り込みについても、それぞれの割り込みルーチンの先頭で、優先順位のフラグを制御すれば可能です。

ではアドレスが 0038h に固定されているモード 1 の場合はどうすればよいでしょうか。これも基本的な考え方は同じです。図 14 に割り込みモード 1 における多重割り込み制御の流れを示します。

モード 1 ですから、タイマ割り込みもプリンタの割り込みも、すべて同じアドレスにジャンプしてきます。これではプログラムでは何の割り込みが発生したかわからないので、**各割り込み源のステータスをチェックし、何の割り込みが発生したかを調べます。**ステータスを調べる順序は優先順位の高い順から調べます。

### モード 2 割り込み

#### ● モード 2 割り込みとは

モード 2 割り込みは Z80 の三つの割り込みモードの中でもっとも強力なモードです。

割り込み処理ルーチンは最大 128 種類までハードウェア的に分岐することが可能です。

分岐の方法としては、割り込みが発生したコントローラが、CPU が割り込みベクタ要求を発生するタイミングで、1 バイトの割り込みベクタをデータ・バスに出力することで実現します。この動作そのものはモード 0 やモード 1 と同じです。

図 15 にモード 2 割り込みの割り込み処理ルーチンへの分岐手順を示します。



〈リスト 3〉 INT 割り込み (モード 1) で LED を反転表示するプログラム

START:	org	0000h	; スタート
	ld	sp, 0	; スタック設定
	ld	a, 0	
	im	1	; 割り込みモード 1
	ei		; 割り込み受け付け許可
LABEL1:	jp	LABEL1	; 無限ループ
INTTOP:	org	0038h	; RST 38H モード 1 割り込み
	xor	02h	; LED 点灯 (20H), a
	out		; 割り込み受け付け許可
	ei		; メイン・ルーチンへ戻る
	reti		

モード 0 割り込みの RST 命令と比べると、分岐できるアドレスが自由になり、しかも 128 種類の割り込みに対応できるなど、かなり強化されています。

また割り込みベクタ・テーブルを RAM 領域に設定することも可能で、割り込み処理ルーチンのアドレスの書き換えをプログラムで行えば、128 種類以上の分岐も可能となります。

#### ● デイジィ・チェーンによる割り込み優先順位

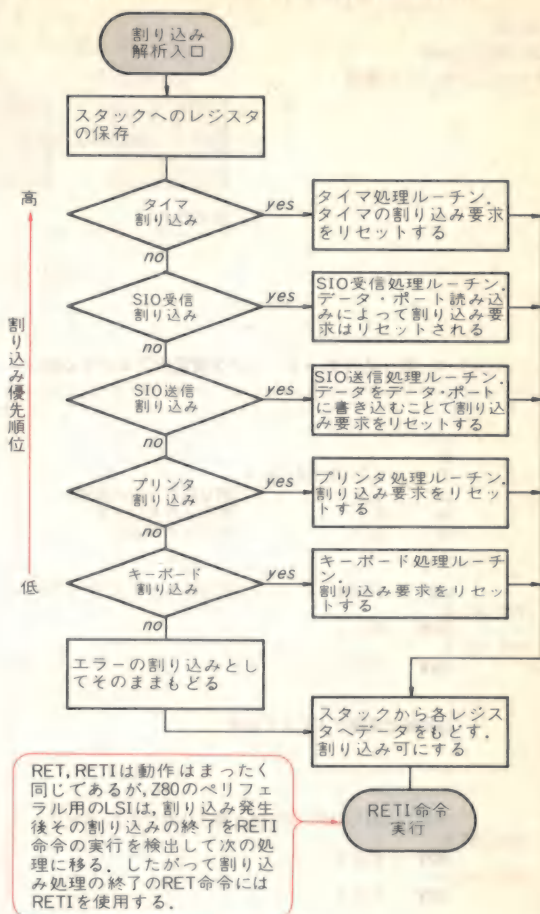
モード 2 割り込みのさらなる特徴として、**割り込み優先順位をハードウェアで決定**できることがあげられます。

Z80 周辺 LSI では、デイジィ・チェーンという手法でこの優先順位の管理を実現しています。これは図 16 に示すように割り込みを発生させる LSI の制御端子を、割り込みの優先順に直列に接続し、自分より下位の LSI に対し割り込み禁止を伝達します。

Z80 周辺 LSI には IEI (Interrupt Enable In) と IEO (Interrupt Enable Out) という二つの端子が用意されており、これらを接続します。

最も優先順位の高い LSI の IEI は +5 V に接続します。各 LSI の伝搬遅延時間の関係で、あまり多くの LSI を接続できません。規格上は四つまでとなってい

〈図 14〉 モード 1 における多重割り込みの流れ



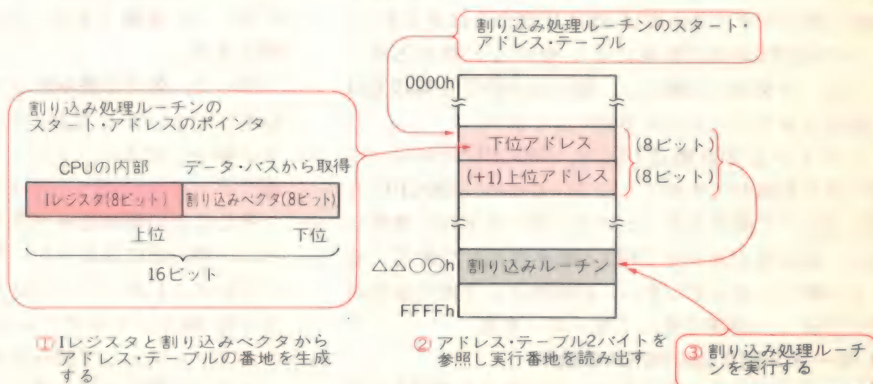
ます。外付け回路を設けることで、より多くの LSI を接続することも可能です。

#### ● Z80 周辺 LSI とモード 2 割り込み

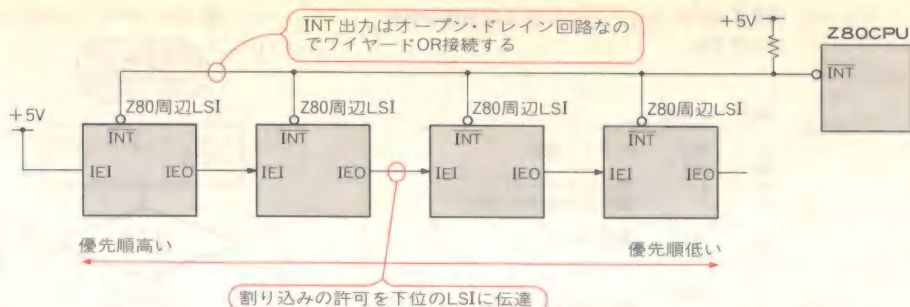
さらに Z80 周辺 LSI は、割り込み処理終了の情報を、周辺 LSI がデータ・バスを常時監視し判断することで、自動的に割り込み要求をとりさげるように設計されています。この割り込み処理終了を周辺 LSI に

〈図 15〉

割り込み処理ルーチン  
実行の手順



〈図 16〉  
Z80 周辺 LSI の  
ティジ・チェーン接続



〈リスト 4〉 割り込みモード、ベクタ設定のプログラム例

```

DI
LD  A, HIGH PIO_INT_A
LD  L, A          ;割り込みベクタ設定
IM  2            ;割り込みモード 2
LD  SP, 0         ;スタックの設定

ORG  1000H        ;割り込みベクタ・テーブル
PIO_INT_A:
DEFW PIO_A       ;
PIO_INT_B:
DEFW PIO_A       ;

(a)ORGで直接アドレスを指定

ORG  ($+1) AND 0FFFh ;割り込みベクタ・テーブル
PIO_INT_A:
DEFW  PIO_A       ;
PIO_INT_B:
DEFW  PIO_A       ;

(b) 算術演算子などで偶数アドレスに自動設定

```

〈リスト 5〉 割り込みベクタ・テーブルのアドレスに注意

```

DI
LD  A, HIGH PIO_INT_A
LD  L, A          ;割り込みベクタ設定
IM  2            ;割り込みモード 2
LD  SP, 0         ;スタックの設定

ORG  ($+1) AND 0FFFh ;この結果が1FFFhのとき

PIO_INT_A:
DEFW  PIO_A       ;1FFFhに格納される
PIO_INT_B:
DEFW  PIO_A       ;2000hに格納される

END

```

明示するために、**RETI 命令**があります。

RETI 命令は、周辺 LSI に対して割り込み処理ルーチンの終了を知らせる役目があり、Z80 周辺 LSI でモード 2 を使う場合には非常に重要な命令ですが、プログラマ的には単なるサブルーチンから復帰する RET 命令と同じです。この点は RETN 命令とは逆です。

この RETI 命令の監視により 8259 など割り込みコントローラを使った場合の、割り込み終了を周辺 LSI に設定するプログラムが不要になります。

このように Z80 周辺 LSI は、Z80CPU のモード 2 割り込みを強力にサポートしているため、Z80CPU と組み合わせて使用することでその能力を十分に発揮します。逆にそのほかの CPU と組み合わせて使用することが難しくなってしまう、汎用性という点では他の周辺 LSI に一步譲る感じになっています。

#### ● モード 2 割り込みのバスの動作

モード 2 割り込み発生時のバスの動作を図 17 に示

します。INT 割り込み認識後、モード 0 やモード 1 と同様、7クロックのスペシャル M1 サイクルを実行します。

スペシャル M1 サイクルでは  $\overline{M1}$  と  $\overline{IORQ}$  が L レベルになることで、CPU が割り込みベクタを読み込むわけですが、この割り込みベクタは、モード 0 のような Z80 が実行できる命令ではありません。いわゆる **ベクタ・アドレス** になります。

CPU は図 17 のように割り込みを発生したデバイスから、割り込みベクタと呼ばれるデータを読み取ると、割り込み処理が終了して再び本来実行していたプログラムに復帰できるように、PC の値をスタックに積みま

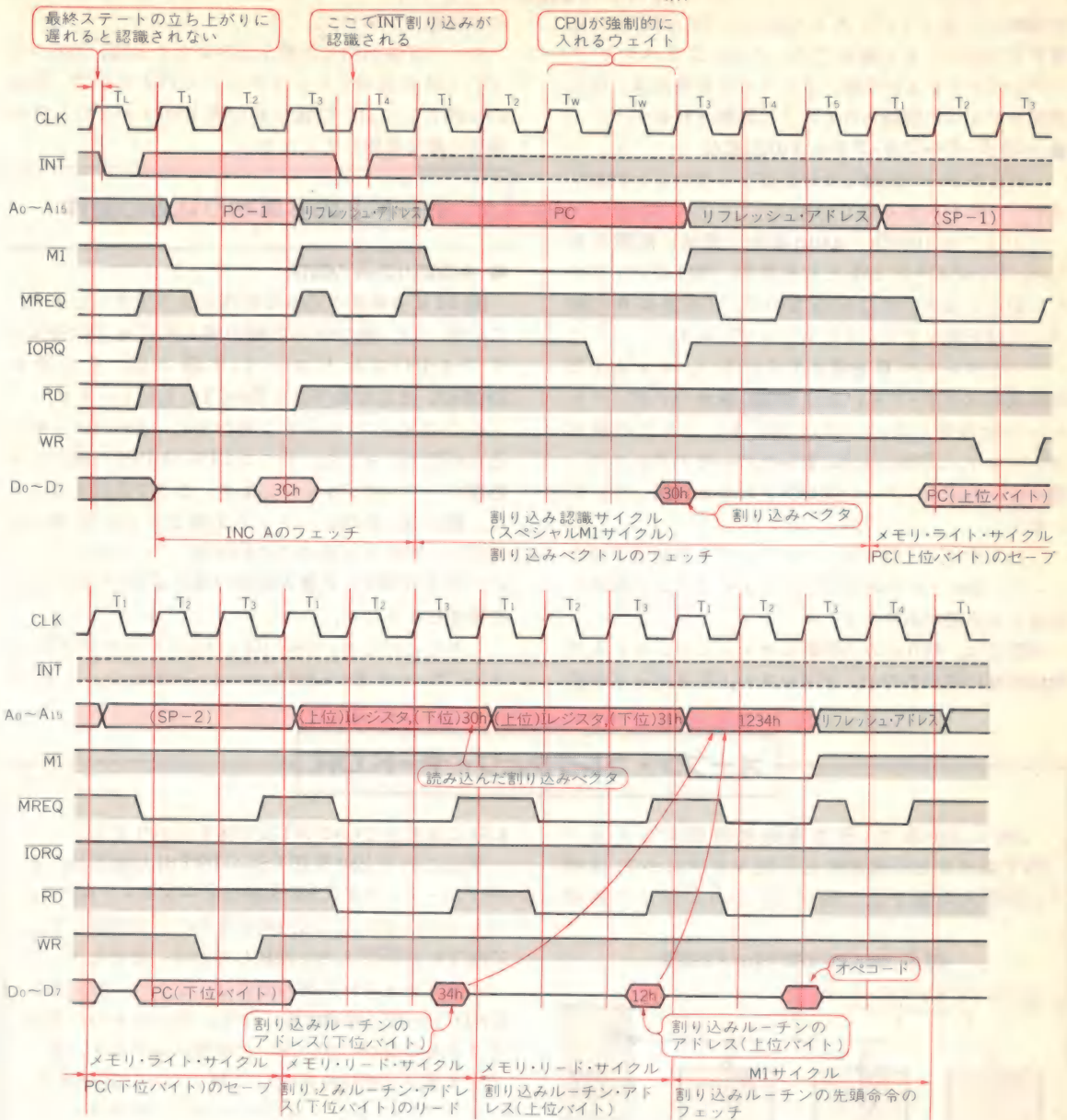
す。そして、先ほど読み取った割り込みベクタの値を下位 8 ビット、そして I (インタラプト・ベクタ) レジスタの値を上位 8 ビットとした 16 ビットのアドレス、**割り込みベクタ・テーブル・アドレス** を生成します。

そしてこの割り込みベクタ・テーブル・アドレスのメモリの値を下位 8 ビット、割り込みベクタ・テーブル・アドレス+1 のアドレスのメモリの値を上位 8 ビットとした 16 ビットのアドレスを、**割り込み処理ルーチンのアドレス** として読み込みます。

こうして割り込み処理ルーチンの先頭にジャンプし、



〈図 17〉 モード 2 割り込みのバスの動作



割り込みプログラムが動き出します。

割り込み処理プログラムが終了したら、次の割り込みに備えて割り込み受け付けを許可するために EI 命令を実行し、割り込み発生前のアドレスに復帰するために RETI 命令で戻ります。これがモード 2 割り込みの大まかな流れです。

〈末木 豊〉

### ● 割り込みモード 2 の割り込みプログラム

まず**割り込みベクタ・テーブルをどこに配置**するかという問題があります。16 ビットのアドレスのうち、上位 8 ビットは I レジスタで、下位 8 ビットは割り込みを発生したデバイスが出力するので、**基本的には**

64 K バイトの空間の好きな部分に配置することができます。

リスト 4 (a)に一般的なモード 2 割り込み処理のベクタ設定と初期化ルーチン例を示します。この例のように、ORG 命令で任意のアドレスに指定する方法もあります。しかしこの例では、プログラムの容量が増加したとき、割り込みベクタ・テーブルにプログラム領域が重なってしまうかもしれません。

当然、そのときは割り込みベクタ・テーブルの ORG の値をずらせばすみますが、いちいちプログラムがどのアドレスまで使うかを気にしなければならないとい

うのも不便です。そこでロケーション・カウンタと算術演算子を使って、リスト4(b)のように自動的に配置する方法をとると便利です。これによりベクタ・テーブルのアドレスが奇数アドレスになる場合は、次の偶数アドレスに格納されるように調整されます。

### ● ベクタ・テーブル・アドレスの注意点

しかしもう一つ注意しなければならないことがあります。リスト5をみてください。

このような1FFEh~2001h番地の領域に配置されると、Iレジスタは1Fhとなります。PIOのチャンネルAに設定するベクタはFEhなので、チャンネルAの割り込みは正常なアドレスにジャンプします。

ここでチャンネルBを考えてください。チャンネルBの本来のベクタ・アドレスは2000h番地ですが、チャンネルBに設定したベクタは00h、Iレジスタの値が1Fhとなり、1F00h番地に書かれているアドレスを、割り込み処理ルーチンの開始アドレスと認識してしまいます。

このように、上位8ビットがIレジスタの値で決まるので、256バイト単位で切りのよいアドレス領域に配置する必要があります。

実際には、割り込みの個数はほとんどのシステムで数個程度に収まるので、アドレスの上位8ビットが変

化しない数十バイトの連続したメモリ領域が確保できれば問題ありません。

モード2を活用した割り込みシステムは、Z80ファミリ LSI を活用したシステムともいえるので、具体的な例としては、以後の章の各 Z80 ファミリ LSI の活用の章を参照してください。

## モード2の多重割り込みの動作

### ● 多重割り込みの動作

図18に多重割り込みの動作のようすを示します。この例では、割り込み処理の優先順位を Z80SIO(シリアル I/O コントローラ)が最上位、その次に Z80PIO、そして最下位を Z80CTC としています。

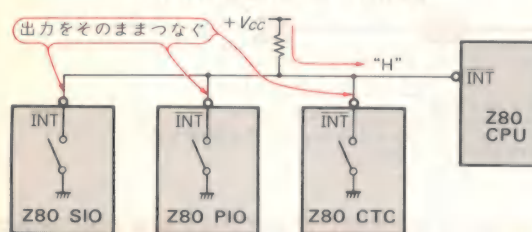
ここでメイン・ルーチン実行中に、Z80PIOの割り込みが発生しました。そこで CPU は PIO の割り込み処理ルーチンにジャンプします。ここで注目することは、割り込み処理ルーチンの先頭ですぐに **EI 命令** を実行し、**割り込み受け付け**を許可している点です。これでは Z80PIO より優先順位の低い CTC が割り込みを発生しそうです。

しかしこのとき、**Z80PIO の IEO**(インタラプト・イネーブル・アウト)が**Lレベル**になり、これが

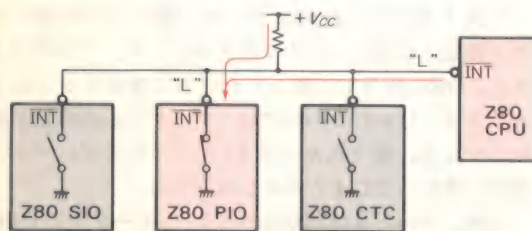
## オープン・コレクタとワイヤード OR

Z80にZ80周辺LSIを複数個接続するとき、INT端子をどう処理するか悩みませんか？ Z80にINT入力端子は1個だけ、それに対して周辺

〈図A〉 Z80CPUの割り込み入力回路



(a) 通常状態



(b) 割り込み発生

LSIにはそれぞれにINT出力端子があります。

実はこれら Z80 周辺 LSI の INT 出力端子は、オープン・コレクタ (CMOS 版はオープン・ドレイン) になっているので、出力同士をそのまま接続して、Z80CPU の INT 入力に接続してかまいません。

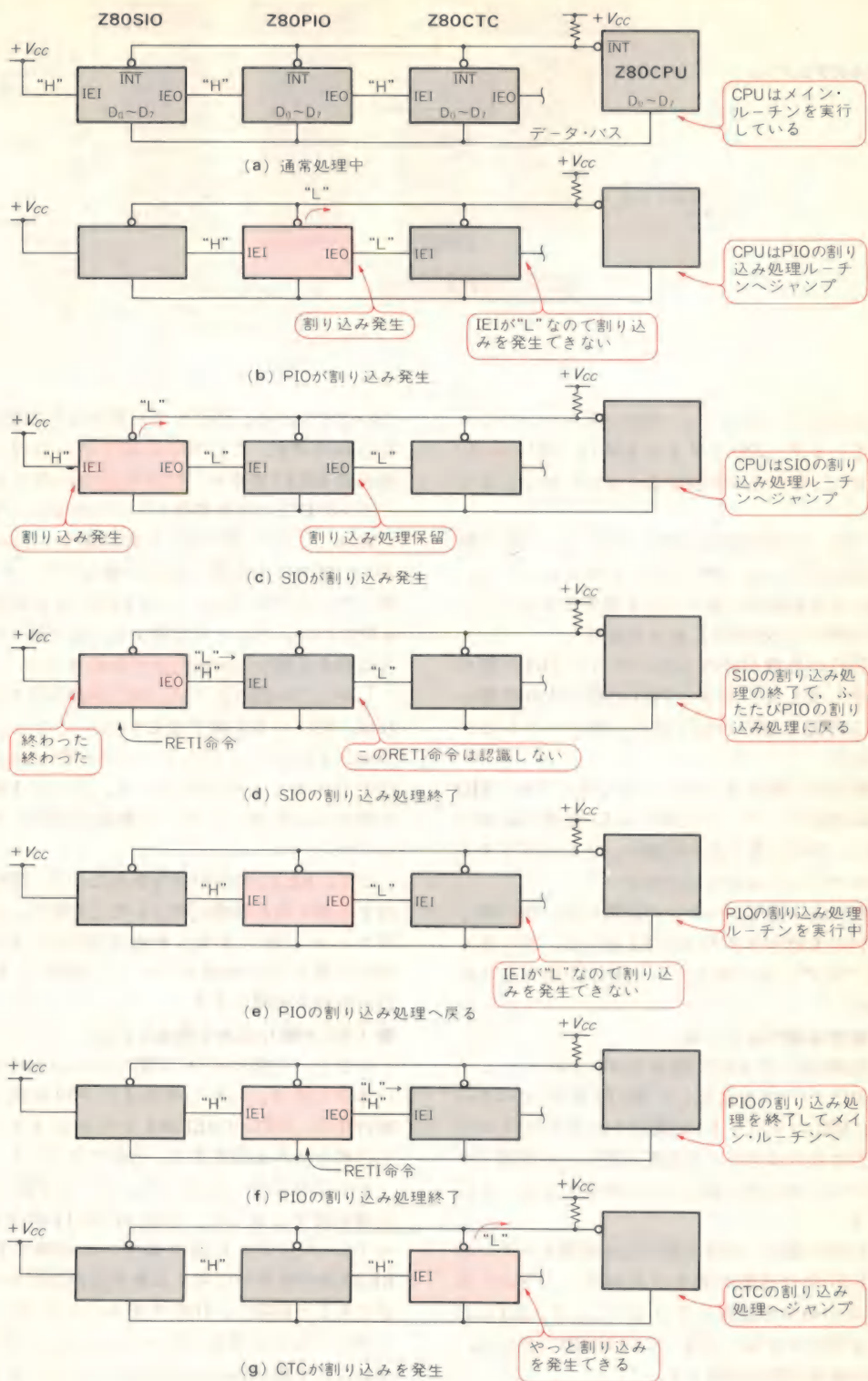
オープン・コレクタの出力は、Lレベルのときは GND との間が導通状態になり、Hレベルのときはどこともつながらず、浮いた状態になります。そこでプルアップ抵抗を接続して Hレベルに定めます。

オープン・コレクタはライン・ドライバやリレー駆動など、インターフェース回路によく用いられます。ダーリントン・トランジスタ・アレイもその仲間です。汎用ロジック IC としては、LS05、LS06、LS07 などがオープン・コレクタ出力です。

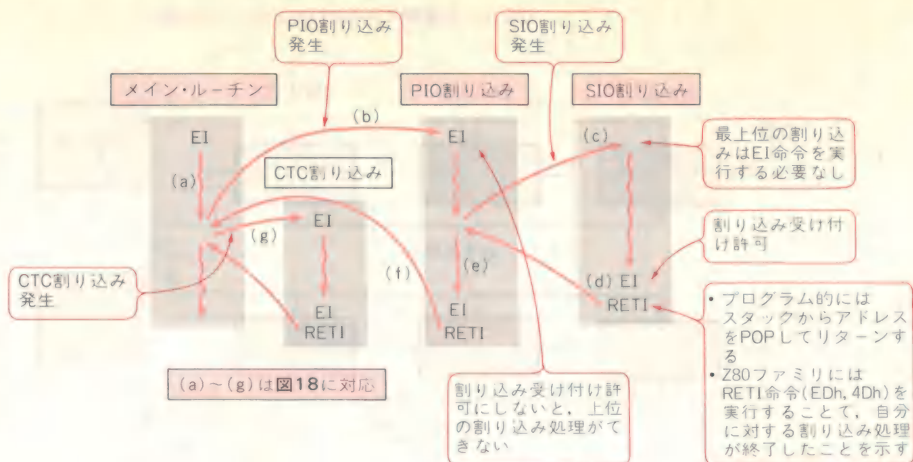
オープン・コレクタは自分自身から Hレベルを出しませんから、出力同士をそのまま接続できます。その場合、どれかの出力が Lレベルになれば共通出力が Lレベルになるわけで、負論理の OR 回路と同じ動作になります。これを特にワイヤード OR 回路と呼び、リセット回路や割り込み回路など不特定個数の信号源を想定した回路に用いられます (図A)。



〈図 18〉 多重割り込みのハードウェア的な動作



〈図 19〉  
多重割り込みのプログラ  
的な動作



Z80CTCのIEI(インタラプト・イネーブル・イン)にも接続されています。Z80ファミリLSIは、IEIがLレベルになると、割り込みを発生できない仕組みになっています。

よってPIOより優先順位の低いCTCは、PIOの割り込み処理が終わるまで割り込みを発生しません。PIOの割り込み処理中に割り込みを発生できるデバイスは、この例ではZ80SIOしかありません。

さて、PIOの処理が終わらないうちにSIOが割り込みを発生しました。するとPIOの割り込み処理ルーチンから今度はSIOの割り込み処理ルーチンにジャンプします。

SIOが最上位の割り込みデバイスであるため、SIOの割り込み処理ルーチンの先頭ではEI命令の必要はありません。SIOの割り込み処理ルーチンが終了するまでほかのデバイスは待つだけです。

SIOの割り込み処理ルーチンの最後には、次の割り込み受け付けを許可するためのEI命令と、割り込み発生前のプログラムに戻るためにRETI命令を入れておきます。

#### ● RETI命令は誰がみている？

さて、問題はこのRETI命令です。Z80ファミリLSIは、データ・バスを監視してRETI命令(オペコードはEDh 4Dhの2バイト)が実行されたかどうかを調べ、実行されればデバイス内部の割り込み制御フラグをクリアし、次に再び割り込みを発生できるように準備します。

ではこの例の場合、SIOの割り込み処理ルーチンの最後のRETI命令は誰が見ているのでしょうか。当然SIOはこの命令を監視しているでしょう。CTCは割り込みを発生できないで待っている状態ですから、RETI命令認識以前の問題です。

問題はPIOです。このSIOの割り込み処理が発生する前は、PIOの割り込み処理ルーチンを実行してい

たわけですから、PIOもまだ割り込み処理が終わっていないわけです。このSIOの割り込み処理ルーチンの最後のRETI命令を、PIOが自分の割り込み処理ルーチンの終了だと誤認識してしまわないのでしょうか。

実はここにもIEI端子とIEO端子がかかわってきます。確かに多重割り込みの場合、データ・バスを監視しているだけでは、そのRETI命令が自分に対する物なのか、それとも自分より上位の割り込みデバイスに対する物なのか判断はできません。

しかしこの例のように、SIOが割り込み処理中であれば、PIOのIEI端子はLレベルになっています。つまりPIOより上位のデバイスが割り込み処理を実行中であることがわかります。よってPIOは、SIOの割り込み処理ルーチンの最後のRETI命令は認識しません。

さて、RETI命令が実行されたので、SIOは自分に対する割り込み処理が終了したと判断し、IEO端子をHレベルに戻します。すると同時に、CPUは再びPIOの割り込み処理ルーチンに復帰し、PIOの割り込み処理を再開します。

#### ● CTCが割り込みを発生したい

ここで、今度はCTCが割り込みの処理を要求しているとします。しかし今はまだPIOが割り込み処理実行中で、CTCのIEIはLレベルのままです。これでは割り込みを発生することができません。

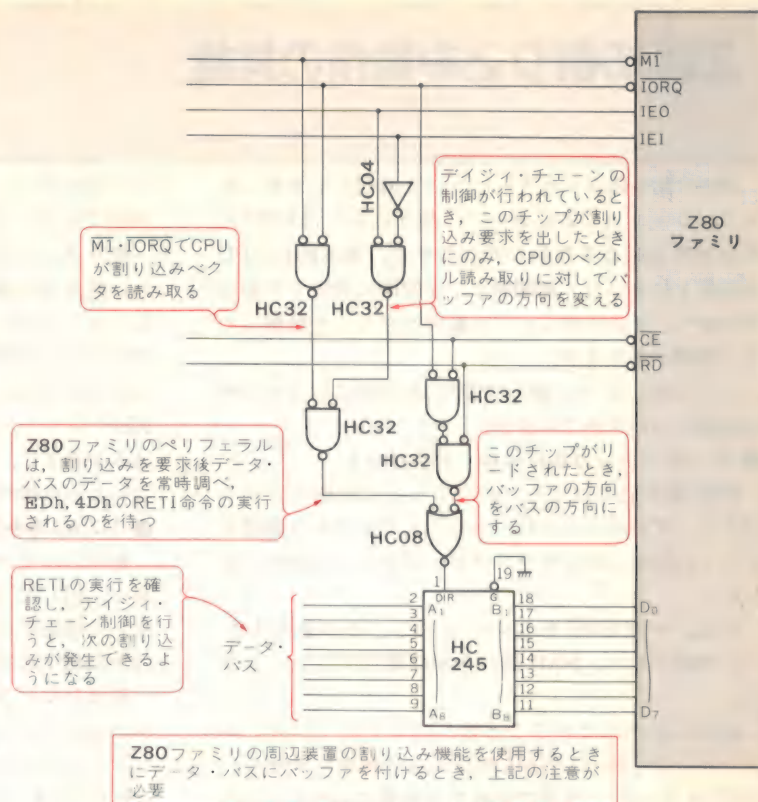
そうこうしているうちに、やっとPIOの割り込み処理が終了しました。CPUはRETI命令を実行してメイン・ルーチンに戻ります。と同時にPIOはそのRETI命令を自分に対する割り込み処理ルーチンの終了であると認識し、IEOをHレベルに戻します。

そして自分のIEIがHレベルになったCTCは、やっとここで割り込みを発生することができるのです。図18の動作をプログラムのみたのが図19です。よく見比べてみてください。



〈図 20〉

割り込み機能使用時のバッファ・コントロール



## ● EI 命令と RETI 命令

ここで、各割り込みルーチンの最後の EI 命令や RETI 命令の動作について確認します。

EI 命令は割り込みを許可する命令です。これを実行しないと、いくらデバイスが割り込みを発生して INT 端子を L レベルにしても、CPU はこれを無視して処理を続けます。

RETI 命令は、CPU についてだけみると、通常の RET 命令の動作となんら変わりません。スタックに積まれたアドレスを POP して、そのアドレスに戻るだけです。

しかし RETI 命令は、Z80 ファミリー LSI にとっては、自分が発生した割り込みのルーチンの終了を判断する大切なキーワードとなります。

もしコーディング時のミスで、RETI 命令を単に RET と記述してしまっても、CPU 単体としての動作は RETI も RET も同じですから、割り込み処理ルーチンを終了して元のアドレスに復帰します。

しかし、Z80 ファミリーは RETI 命令が実行されていないので、自分の割り込み処理ルーチンが終了したと判断できず、CPU がメイン・ルーチンに復帰しても、自分の割り込み処理ルーチンを続けていると認識し、次の割り込みを発生しません。

またデジィ・チェーン接続で多重割り込みができ

る場合は、そのデバイスより優先順位の低いデバイスも、割り込みを発生できなくなってしまう。

## ● CPU と Z80 ファミリー LSI 間にバッファを入れた場合

最後に、Z80CPU に多数の周辺 LSI を接続したときの注意点です。Z80 ファミリーにかぎらず、ロジック IC は一つの出力からいくつもの IC の入力へは接続できません。そのため、多数の I/O を増設する場合など、**バスの駆動能力が低下**し誤動作の原因になります。

そこでデータ・バスやアドレス・バスに、HC245 などの**バッファ IC**を接続し、多数の周辺 IC を接続しても CPU がすべての IC に正しくデータが送られるようにします。そして CPU がデータを読み込むときは外から CPU へデータが向かう方向へ、書き込むときは CPU から外へ向かう方向に制御します。

ここで注意点があります。これまで説明したように Z80CPU とそのファミリーは、割り込みベクタや RETI 命令の認識など、単純なデータの入出力以外にも情報のやり取りをしています。このため HC245 などのバッファを間に入れる場合は、図 20 のように  $\overline{\text{IORQ}}$  や  $\overline{\text{MI}}$ 、 $\overline{\text{IE0}}$  や  $\overline{\text{IE1}}$  などの信号の状態をも考慮して、**バッファの方向を制御してやる必要があります**。

〈野口智樹〉

(トランジスタ技術1994年6月号および10月号に加筆、修正)

## Z80 の割り込み動作の詳細

磯沼 薫

Z80 の割り込みシステムについて説明してきました。マイコン・システムにおいて、割り込みとは CPU の利用効率を高める有効な方法ですが、基本的に CPU の動作とはまったく無関係に、非同期に発生するわけですから、そのタイミングと動作をきちんと理解しておく必要があります。

ここでは、さらに細かな割り込み時における Z80 の挙動についてみてみます。

### ● エッジ・トリガかレベル・トリガか？

$\overline{\text{NMI}}$  割り込みは立ち下がりエッジで内部のフリップフロップがセットされ、マシン・サイクルの最終ステートの立ち上がりでサンプリングされると説明しました。

では、一度  $\overline{\text{NMI}}$  を L レベルにし、その後も L レベルを保ったら、 $\overline{\text{NMI}}$  割り込みは発生するでしょうか。

常に L レベルということは、立ち下がりエッジがないわけですから、 $\overline{\text{NMI}}$  割り込みは発生しません。 $\overline{\text{NMI}}$  はエッジ・トリガであることを頭に入れてください。

では、 $\overline{\text{INT}}$  はどうかというと、こちらはレベル・ト

リガですから、 $\overline{\text{INT}}$  端子を常に L レベルにすると、何度でも割り込みが発生します。

割り込みが受け付けられるためには EI 命令を実行し、割り込み受け付け許可状態にしないとはなりません。よって割り込みルーチンの最後には EI 命令と RETI 命令を置きます。割り込みが実際に受け付けられるのは EI 命令の次の命令を実行してからですから、REIT 命令でメイン・ルーチンに戻ったとたんに割り込みが発生し、メイン・ルーチンがちっとも実行されないという状況に陥ります。

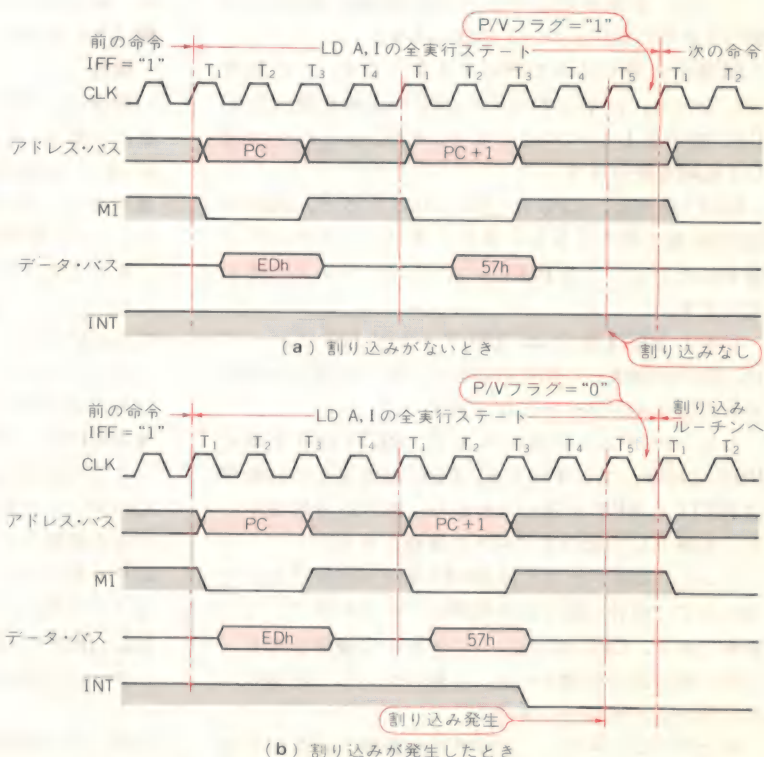
### ● LD A,I 命令実行時に割り込みが発生したら？

割り込み受け付け禁止状態か許可状態かを調べる命令に、LD A, I 命令があることはすでに説明しました。では割り込み受け付け許可状態で、この命令を実行中に割り込みが発生したらどうなるのでしょうか。

実はなんと NMOS 版の Z80 では、実行中に割り込みが発生すると、割り込み受け付け禁止中であるがごとくフラグがリセットしてしまう (P/V フラグが 0 になる) のです。すべてについて確認したわけではありませんが筆者が実験した限りでは、どのメーカー製でも NMOS 版ではこのようになるようです (図 A)。

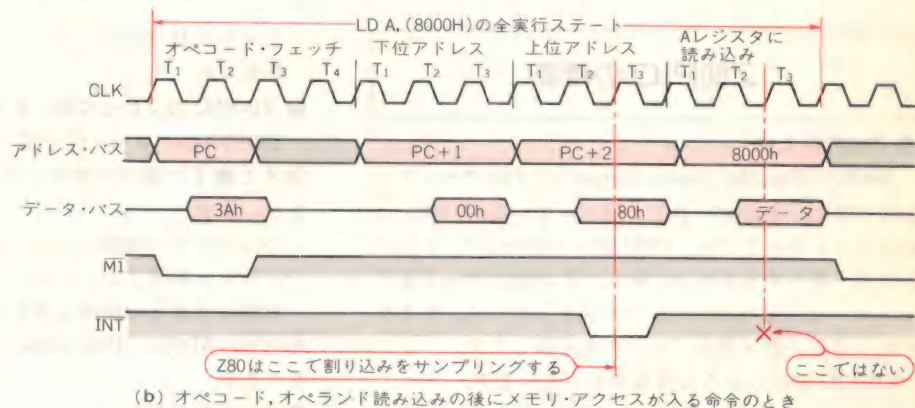
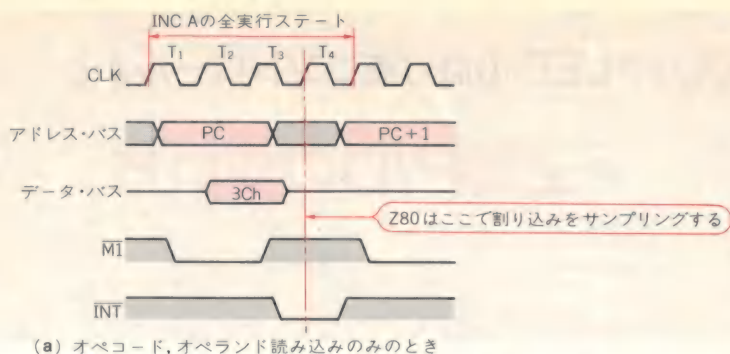
〈図 A〉

LD A, I 命令実行中に割り込みが発生すると





〈図 B〉  
Z80のINT割り込み受け付けタイミング



ではCMOS版ではどうかというと、サイログのZ80や周辺LSIを集積したZ84C15/C11などではこのような誤動作はありません。割り込み受け付け許可状態であれば、必ずP/Vフラグがセットされます。

しかし同じCMOS版といえども、東芝製のTMPZ84C015では誤動作しました。

また日立の64180についても、R1およびZバージョンからはマニュアルに「LD A, I および LD I, A 命令実行中は、割り込みをサンプリングしません。」と書かれています。

最後に最新のCPUである川崎製鉄のKL5C8012ではどうかというと、これも命令実行中に割り込みが発生しても誤動作はしないようです。

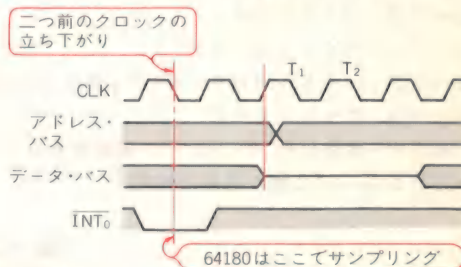
### ● 割り込みはどこでサンプリングされる？

Z80の割り込みは命令の最終ステートの立ち上がりでサンプリングされるとありますが、本当でしょうか？

Z80の割り込み受け付けの瞬間のバスのようすを図Bに示します。図B(a)のようにオペコードを読み込んで内部で実行するだけの命令のときはたしかにそうなのですが、図B(b)のようにメモリの読み書きなどを伴う命令のときは、最後ステートではなく、**オペランドの読み込みの最終ステートの立ち上がりでサンプリング**されるようです。

ちなみに64180ではこのような命令のときも、割り

〈図 C〉 64180の割り込み受け付けタイミング



込みのサンプリング・タイミングは最終ステートの2クロック前の立ち下がりとなります(図C)。

### ● NMI と DMA

NMIは電源の異常やメモリのパリティ・エラーなど、エラーや緊急事態の割り込みに使うべきで、リセットに次ぐ最上位の割り込みだと説明しました。

しかしZ80では、NMIとDMAとでは**DMAのほうが優先順位が高く**、DMAがハングアップするとZ80CPUは手も足も出せなくなります。

64180ではNMIはDMAよりも優先順位が高く、NMI割り込みでDMAを中止させるようなことも可能です。

もっとも最近ではZ80にDMAはほとんど使用しないので、あまり考える必要はないと思います。

## スイッチ入力やLEDの点灯をコントロールする

### Z80PIOの使い方

野口智樹

#### Z80PIO の概要

##### ● Z80PIO とは

Z80PIO (Parallel Input Output) は Z80 ファミリの LSI で 8 ビットのポートを 2 個もっている、**パラレル I/O コントローラ**です。CPU 側から見たとき、2 個の I/O ポート・チャンネル A、B と、その動作モードをコントロールするためのレジスタがチャンネル A、B それぞれにあり、計 4 個のアドレスを占有します。

Z80PIO は次のような特徴をもっています。

- ・チャンネル A/B という二つの 8 ビット・ポート
- ・各チャンネルとも以下の**四つのモード**から選択してモードをプログラムできる
  - モード 0: バイト出力モード
  - モード 1: バイト入力モード
  - モード 2: バイト入出力モード (チャンネル A のみ)
  - モード 3: ビット・モード
- ・A、B 各チャンネルに**ハンドシェイク機能**がある
- ・Z80CPU のモード 2 割り込みに対応

・チャンネル B 出力は、ダーリントン・トランジスタを駆動可能

##### ● Z80PIO のブロック図、ピン配置

Z80PIO のブロック図を図 1 に、ピン配置を図 2 に、図 3 と表 1 に電気的特性と入出力端子のタイミングを示します。

Z80CPU との接続については、第 4 章で説明したのでそちらを参照してください。

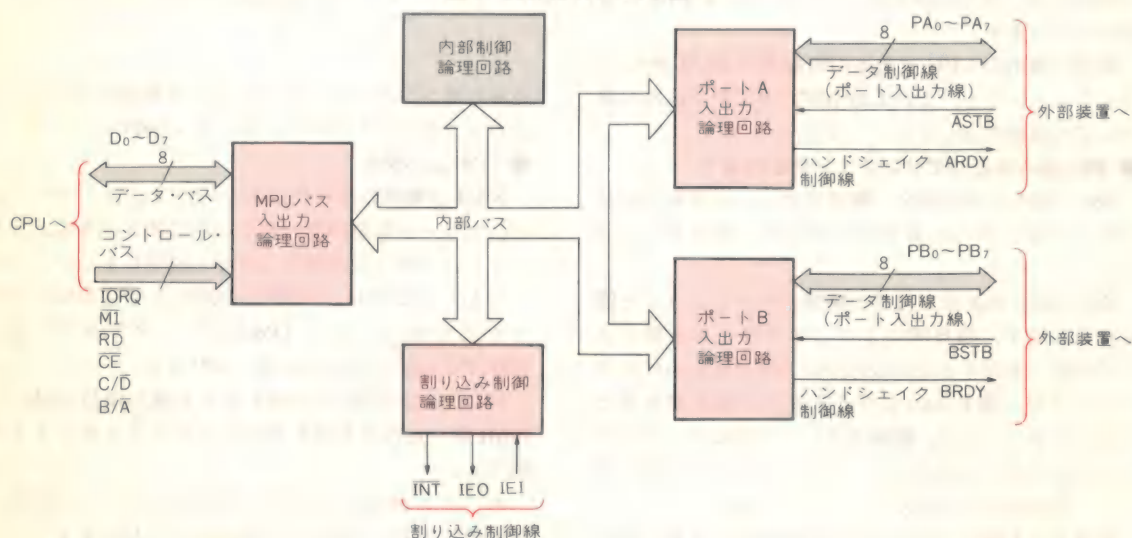
実際に外部との接続に使われる端子は、 $PA_0 \sim PA_7$ 、 $\overline{ASTB}$ 、 $ADRY$ 、 $PB_0 \sim PB_7$ 、 $\overline{BSTB}$ 、 $BRDY$  端子となります。

##### ● Z80PIO のリセット

Z80PIO には IC パッケージのピン数の制限から、**RESET 入力端子がありませんが**、電源投入時に自動的に内部をリセットする、**パワー ON リセット機能が内蔵**されています。また第 4 章の p.56 ですでに説明したような外部回路を接続すれば、任意にリセットをかけることもできます。

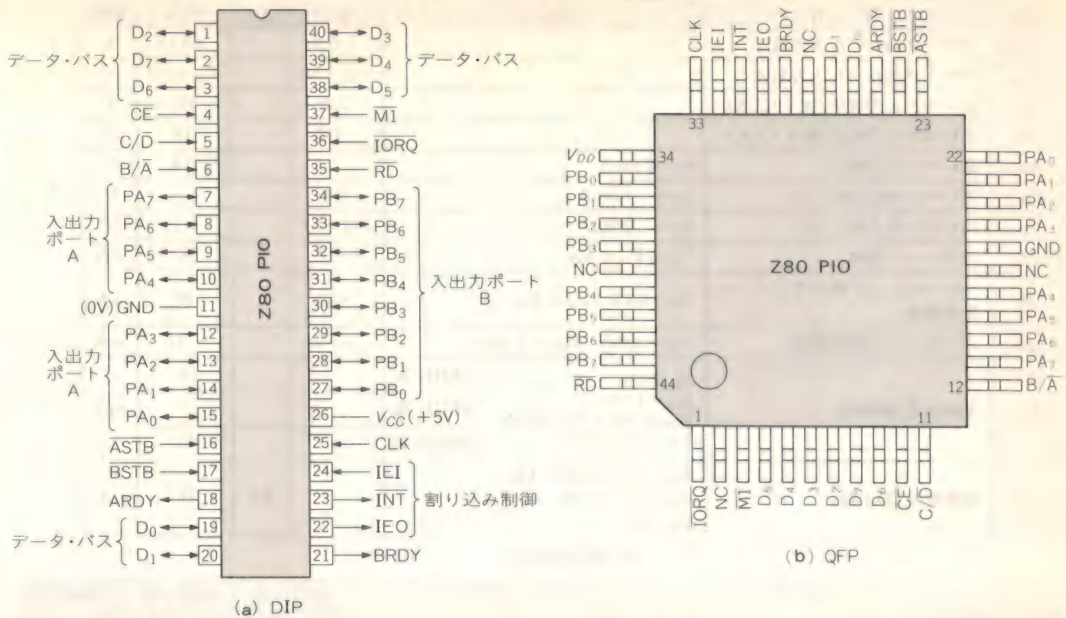
また Z80PIO はリセットされるとポート A、B とも、次の状態になります。

〈図 1〉 Z80PIO のブロック図





〈図2〉 Z80PIO のピン配置



- ・割り込み禁止
- ・モード1(入力)に設定
- ・データ入出力レジスタは全ビットがクリア
- ・マスク制御レジスタは全ビットがセット(マスク)
- ・ポート入出力線はハイ・インピーダンス状態
- ・RDY 端子は L レベル(ノット・レディ)

このリセット状態は、コントロール・ワードの設定があるまで保持されます。

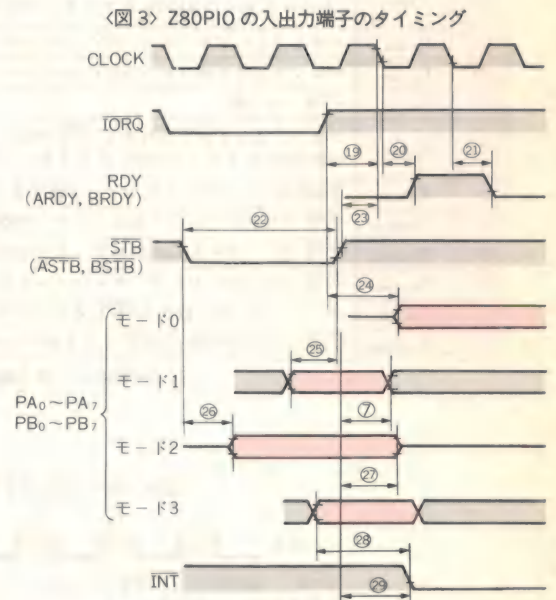
## Z80PIO の各種モード

まず Z80PIO にはどんなモードがあるのか、各モードの動作について説明します。

### ● Z80PIO のモード0(バイト出力モード)

Z80PIO のモード0は**バイト出力モード**です。このモードに設定するとポートの方向は**出力方向**になります。図4にモード0におけるデータ出力のタイミングを示します。PIOの初期設定が終わり、データ・ポートに出力するデータを書き込むと、データ・ポートのデータが出力され、RDY 端子(実際には、チャンネルAを使用するときは ARDY, チャンネルBなら BRDY となる)が H レベルになります。RDY 端子は READY の意味で、準備完了を示す制御線です。

データを受け取る側では、この RDY 端子が H レベルになるのを待ち、H レベルになったらデータを読み込みます。そしてデータを受け取ったことを PIO に知らせるために、 $\overline{\text{STB}}$  端子(実際にはチャンネルAを使用するときは  $\overline{\text{ASTB}}$ , チャンネルBなら  $\overline{\text{BSTB}}$  と



なる)に読み出しクロックを入力します。

相手がデータを受け取らない、つまり  $\overline{\text{STB}}$  端子に読み出しクロックが入力されない限り、データ・ポートには書き込んだデータが出力されつづけ、RDY 端子は H レベルになったままで保持されます。

PIO は  $\overline{\text{STB}}$  端子の立ち上がりエッジを認識すると、RDY 端子を L レベル戻し、また  $\overline{\text{INT}}$  を L レベルにして CPU に対して割り込みを発生します。

この割り込みは、相手がデータを受け取ったという

〈表 1〉 Z80PIO の電気的特性とタイミング

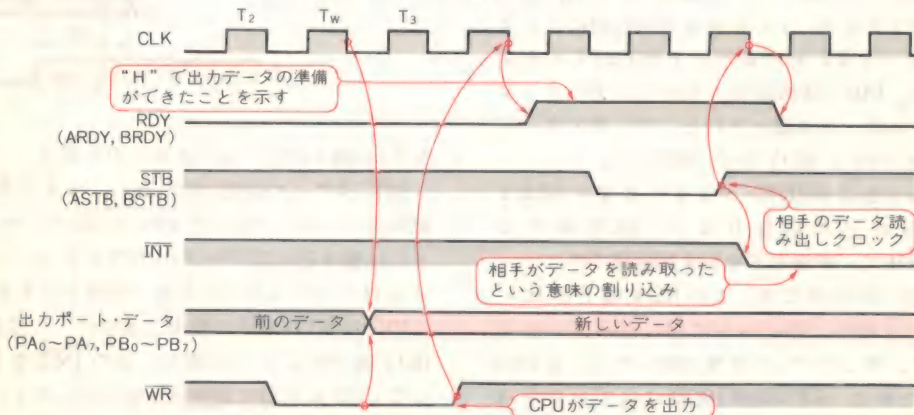
記号	項 目	測定条件	最小	標準	最大	単位
$V_{ILC}$	クロック低レベル入力電圧		-0.3		+0.45	V
$V_{IHC}$	クロック高レベル入力電圧		$V_{CC}-0.6$		$V_{CC}+0.3$	V
$V_{IL}$	低レベル入力電圧(除く CLK)		2.2		$V_{CC}$	V
$V_{IH}$	高レベル入力電圧(除く CLK)		-0.3		0.8	V
$V_{OL}$	低レベル出力電圧	$I_{OL}=2.0\text{mA}$			0.4	V
$V_{OH1}$	高レベル出力電圧(I)	$I_{OH}=-1.6\text{mA}$	2.4			V
$V_{OH2}$	高レベル出力電圧(II)	$I_{OH}=-250\mu\text{A}$	$V_{CC}-0.8$			V
$I_{L1}$	入力リーク電流	$V_{SS}\leq V_{IN}\leq V_{CC}$	-10		10	$\mu\text{A}$
$I_{LO}$	フローティング時の 3 ステート出力電流	$V_{SS}+0.4\leq V_{OUT}\leq V_{CC}$	-10		10	$\mu\text{A}$
$I_{OHD}$	ダーリントン駆動電流	$V_{OH}=1.5\text{V}$ , $R_{EXT}=1.1\text{k}\Omega$	-1.5		-5.0	mA
$I_{CC1}$	電源電流(動作時)	$V_{CC}=5\text{V}$ $f_{CLK}=1/T_{CC}$ $V_{IHC}=V_{IH}=V_{CC}-0.2\text{V}$ $V_{ILC}=V_{IL}=0.2\text{V}$	6MHz 版		6	mA
			8MHz 版		7	
			10MHz 版		12	
$I_{CC2}$	電源電流(静止時)	$V_{CC}=5\text{V}$ , $CLK=V_{CC}$ $V_{IH}=V_{CC}-0.2\text{V}$ $V_{IL}=0.2\text{V}$		0.5	10	$\mu\text{A}$

(a) 電気的特性

番号	記号	項 目	6MHz 版		8MHz 版		10MHz 版	
			最小	最大	最小	最大	最小	最大
19	$T_{CJO(C)}$	クロック立ち下がりに対する $\overline{IORQ}=\text{"H"}$ のセットアップ時間 (次のサイクルで RDY をアクティブにする場合)	170		140		120	
20	$T_{dC(RDY)}$	クロック立ち下がりから RDY 立ち上がりまでの遅延		170		150		130
21	$T_{dC(RDY)}$	クロック立ち下がりから RDY 立ち下がりまでの遅延		120		100		85
22	$T_{wSTB(C)}$	$\overline{STB}$ パルス幅	120		100		80	
23	$T_{sSTB(C)}$	クロック立ち下がりに対する $\overline{STB}$ の立ち上がりのセットアップ時間 (次のサイクルで RDY をアクティブにする場合)	150		120		100	
24	$T_{dIO(PD)}$	$\overline{IORQ}$ 立ち上がりから出力データ確率までの遅延(モード 0)		160		140		120
25	$T_{sPD(STB)}$	$\overline{STB}$ 立ち上がりに対するセットアップ時間(モード 1)	190		140		75	
26	$T_{dSTB(PD)}$	$\overline{STB}$ 立ち下がりからの出力データまでの遅延(モード 2)		180		150		120
27	$T_{dSTB(PdF)}$	$\overline{STB}$ 立ち上がりからデータ・フロートまでの遅延(モード 2)		160		140		120
28	$T_{dPD(INT)}$	ポート・データ一致から INT 立ち下がりまでの遅延(モード 3)		430		360		200
29	$T_{dSTB(INT)}$	$\overline{STB}$ 立ち上がりから INT 立ち下がりまでの遅延		350		290		220

(b) Z80PIO の入出力端子のタイミング(単位: ns)

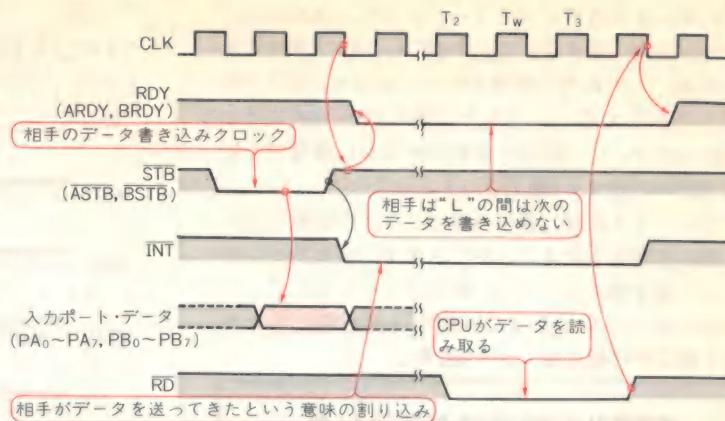
〈図 4〉 モード 0(バイト出力モード)のタイミング





〈図 5〉

モード 1 (バイト入力モード) のタイミング



意味の割り込みと捉えることができます。

### ● Z80PIO のモード 1 (バイト入力モード)

Z80PIO のモード 1 は **バイト入力モード** です。このモードに設定するとポートの方向は **入力方向** になります。図 5 にモード 1 におけるデータ入力のタイミングを示します。PIO の初期設定が終わり、データを受け取れる準備ができると、RDY 端子は H レベル (受け取り準備完了) を示します。

RDY 端子が H レベルであることを確認した相手は、データ・ポートにデータを出力し、データの準備できたことを PIO に示すために、 $\overline{\text{STB}}$  端子に書き込みクロックを入力します。

PIO は、 $\overline{\text{STB}}$  端子の立ち上がりクロックを認識すると、データ・ポートにデータが準備できたとみなし、RDY 端子を L レベルにし、また  $\overline{\text{INT}}$  を L レベルにして **CPU に割り込みを発生** させます。

この割り込みは、相手がデータを送ってきたという意味の割り込みと捉えることができます。

CPU は割り込み処理ルーチンで、PIO のポートからデータを読み込みます。CPU がデータを読み込むと、PIO は RDY 端子を H レベルに戻し、次のデータ入力に備えます。

### ● Z80PIO のモード 2 (バイト入出力モード)

Z80PIO のモード 2 は、**モード 0 とモード 1 を足したような動作** をします。このモードに設定するとポートは **ハイ・インピーダンス** になります。このモードはチャンネル A 専用で、次のような流れで動作するモードです。

#### ▶ PIO 出力

- (1) CPU から PIO のチャンネル A にデータを書き込む。
- (2) そのデータが PIO 内部にラッチされ、レディ (ARDY) が H レベルになる。
- (3) 外部からのストロープ ( $\overline{\text{ASTB}}$ ) 信号が L レベルになると、PIO 内部にラッチされていたデータが、チャンネル A のポートに出力される。

- (4) 外部からのストロープ ( $\overline{\text{ASTB}}$ ) 信号が H レベルに戻ると、その立ち上がりでレディ (ARDY) が L レベルに戻る。このとき、ポートの割り込みが有効な状態ならば、チャンネル A から割り込みが発生する。

#### ▶ PIO 入力

- (5) 外部からデータをチャンネル A のポート上に乗せた状態で、ストロープ ( $\overline{\text{BSTB}}$ ) 信号が L レベルになる。
- (6) チャンネル A のデータが PIO 内部にラッチされる。
- (7) 外部からのストロープ ( $\overline{\text{BSTB}}$ ) 信号が H レベルに戻ると、その立ち上がりでレディ (BRDY) が L レベルになる。このとき、ポートの割り込みが有効な状態ならば、チャンネル B から割り込みが発生する。
- (8) CPU がポートのデータを読み出すと、レディ (BRDY) が H レベルに戻る。

以上のように Z80PIO のモード 2 は、A/B 両チャンネルのレディ (ARDY/BRDY) とストロープ ( $\overline{\text{ASTB}}/\overline{\text{BSTB}}$ ) を使用します。

またチャンネル A がモード 2 のとき、ポート B はモード 3 でしか動作できません。

モード 2 の動作タイミングを図 6 に示します。

### ● Z80PIO モード 3 (ビット・モード)

モード 3 は **ビット・モード** という名前がついているように、ポートの **ビットごとに入力と出力の設定が可能** で、モード 0~2 のような、レディ/ストロープなどのハンドシェイク線を使用しないため、いつでもポートの入出力ができるモードです。

ビットごとに入出力を設定できるといっても、Z80 の I/O 命令は 8 ビットごとにしか入出力できません。このときポートへのリード命令に対して、入力に指定されたビットは PIO のポートの状態を入力しますが、出力と指定されたビットについては、PIO 内部の出力ラッチのデータが読み出され、合成されて 1 バイトのデータとして CPU に送られます。

したがって、ポートの全ビットを出力と指定したと

きに、ポートからデータをリードすると、以前に出力したデータがそのまま入力(リード・バック)されます。

またポートが入力に設定されているとき、**RD 信号の立ち下がり**でポートの入力状態が **PIO 内部にラッチ**されるので、いつ変化するかわからない信号でも安定して読み込めます(図 7)。

またモード 3 には、各信号線のビットの状態により割り込みを発生させることができます。つまりハンドシェイク線を使えなくても、例えばビット 0 とビット 2 が共に H レベルになったら割り込みを発生させるという指定が可能になります(図 8)。

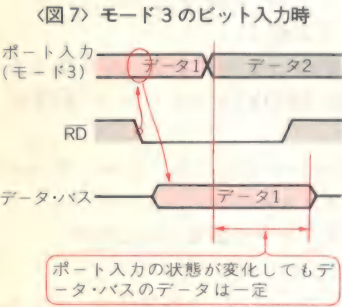
# Z80PIO のプログラミング

## ● PIO のアクセス

Z80PIO は、チャンネル A とチャンネル B の二つのチャンネルがあり、それぞれにデータ・ポートとコントロール・ポートをもっています(表 2)。

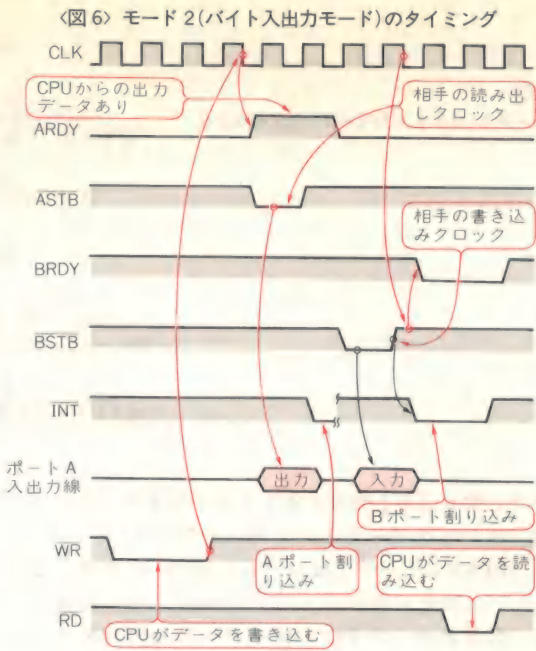
ここで CPU が、チャンネル A のコントロール・ポートに対して書き込みをすると、チャンネル A のモード設定や割り込みベクタの設定などができます。またチャンネル B のデータ・ポートから読み込み動作を行うと、チャンネル B が入力に設定されていればチャンネル B のポートの状態が、出力に設定されていれば書き込んだデータがリード・バックして読み出されます。

図 9 にコントロール・ワードのフォーマットを示します。



〈表 2〉 Z80PIO の各ポートの選択

B/A	C/D	選択されるポート
"L"	"L"	チャンネル A データ・ポート
"L"	"H"	チャンネル A コントロール・ポート
"H"	"L"	チャンネル B データ・ポート
"H"	"H"	チャンネル B コントロール・ポート



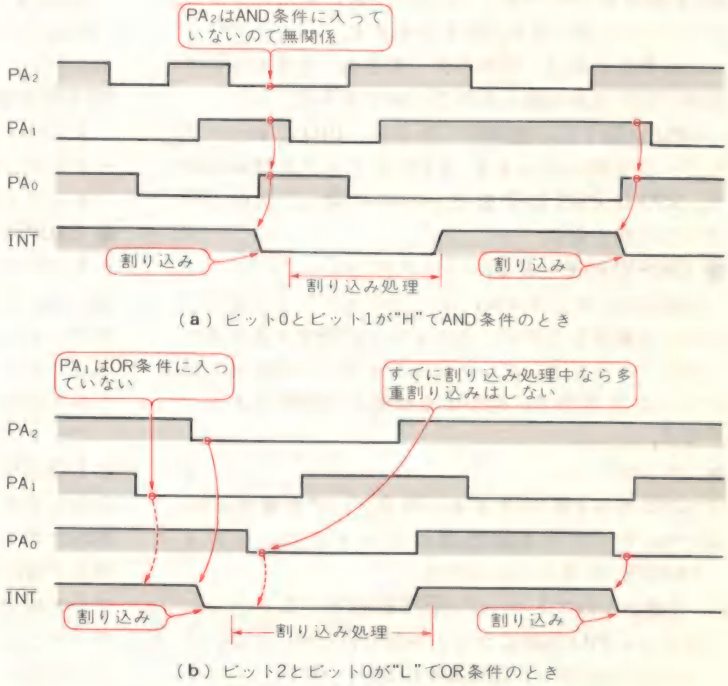
## ● モード 0~2 初期化プログラム

Z80PIO は Z80 周辺 LSI としては比較的単純なものです。それでもモード設定のプログラミングには気を付けなければならないことがあります。

図 10 (a) にモード 0~2 における初期化プログラムの流れを示します。

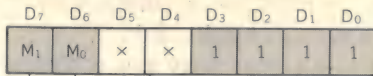
初期化プログラムで注意することは、初期化途中で

〈図 8〉 モード 3 の割り込みの AND, OR 条件



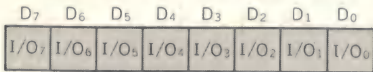


〈図9〉 Z80PIO のコントロール・ワードのフォーマットと初期化の流れ



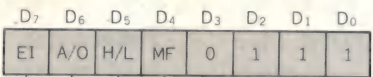
0	0	モード0(出力モード)
0	1	モード1(入力モード)
1	0	モード2(双方向モード)
1	1	モード3(ビット・モード)

(a) モード・ワード



I/O<sub>n</sub>=0 ビットn=出力  
I/O<sub>n</sub>=1 ビットn=入力

(b) データ・ディレクション・ワード



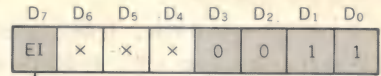
マスク・フォロース(Mask Follows)・・・1で有効

0 : L  
ハイ/ロー(High/Low)1 : H

0 : OR  
アンド/オア(AND/OR)1 : AND

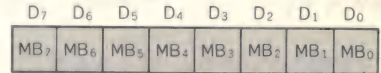
イネーブル・インタラプト(Enable Interrupt)・・・1で有効

(c) インタラプト・コントロール・ワード(その1)



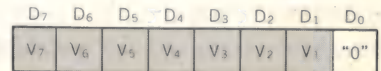
イネーブル・インタラプト

(d) インタラプト・コントロール・ワード(その2)



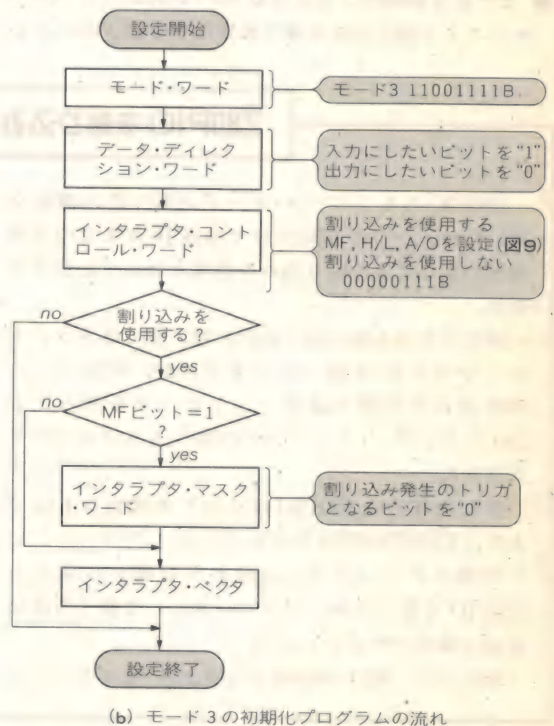
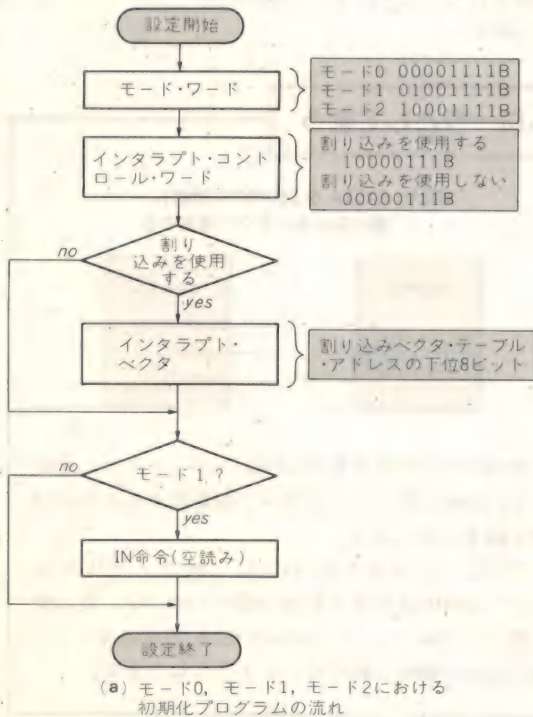
MB<sub>n</sub>=0でインタラプトの対象

(e) インタラプト・マスク・ワード



(f) 割り込みベクタのフォーマット

〈図10〉 PIO の各モードの初期化プログラムの流れ



割り込みが発生しないように、初期化の先頭で割り込み受け付けを禁止する点です。

またモード1の入力モードでは、初期化しただけではRDY端子がHレベルにならず、外部からデータを入力することができません。モード1ではIN命令を入れてRDY端子をHレベルにします。このIN命令のことを空読みとも呼びます。読み込んだデータそのものに意味はないので、データは無視してかまいません。

#### ● モード3初期化プログラム(割り込みなし)

いちばん使用頻度の多いと思われる、単純なビット入出力モードであるモード3です。図10(b)に初期化プログラムの流れを示します。モード3では、ビットごとに入出力の設定ができたので、モード・ワードを書き込んだあとにディレクション・ワードが必要になります。そしてインタラプト・コントロール・ワードの最上位ビットを“0”にして書き込めば、割り込みを使わないモード3の設定が完了します。

モード3で割り込みを使わないといっても、モード初期化の途中で割り込みが発生しないように、割り込み受け付けは禁止します。これは、たとえ割り込みを使わなくても、3バイトのデータを書き込まないとモード3の初期化が終了しないからです。このとき、例えば2バイト目を書き込んだとき割り込みが発生し、その割り込みルーチンでもPIOに対してアクセスがあった場合、書き込みシーケンスの順番がおかしくなってしまう、正常に初期化されません。

#### ● モード3初期化プログラム(割り込みあり)

モード3で割り込みを使う設定は少しやっかいです。

割り込みを使う、使わないの設定は、インタラプト・コントロール・ワードのビット7で指定します。これを“1”にすると割り込みを使用します。

モード0やモード1では、RDY端子やSTB端子を制御することで割り込みを発生しましたが、モード3ではこれらの端子は使用しません。よって割り込みのトリガには、データ・ポート8ビットの状態変化をトリガにします。

そのための指定がビット4～ビット6の指定です。ビット4はMF(マスク・フォローズ)ビットで、このビットを1にすると、次にコントロール・ポートに書き込むデータはインタラプト・マスク・ワードであることを示します。

ビット5は、指定したビットがHレベルになったときに割り込みを発生させるのか、Lレベルになったときに割り込みを発生させるのかの指定です。

ビット6は、インタラプト・マスク・ワードで指定した割り込み発生の対象となるビットが複数の場合、すべての端子がビット5で指定したレベルになったら割り込みを発生させるのか、どれか一つでも指定したレベルになったら割り込みを発生させるのかを指定するビットです。

次は、インタラプト・コントロール・ワードのビット4のMFビットを1にした場合は、インタラプト・マスク・ワードを書き込みます。

これはデータ・ディレクション・ワードで入力に設定したビットのうち、割り込み発生の対象するビットを指定するものです。そして最後に割り込みベクタを設定します。

## Z80PIO を割り込みコントローラとして使う

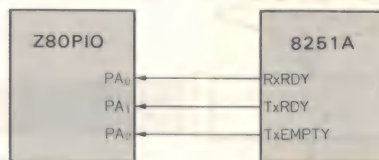
Z80PIOのもつビット・モードの割り込み機能を用いれば、一つのPIOだけで最高16本までの外部信号の変化による割り込みを発生させることができます。

本来ならば多数の割り込みには、割り込みコントローラ用のICを用いるべきですが、8259などの8080系の汎用割り込みコントローラを使ってもZ80のデジィ・チェーン式の割り込みには対応できません。

また現在ではZ80PIOはかなり低価格なLSIであり、TMPZ84C015などのワンチップ・マイコンにも内蔵されています。このような面から見てもZ80PIOを割り込みコントローラとして使う方法は有効な場合が多いでしょう。

例として、8251(8080系のシリアル・インターフェ

〈図A〉 8251AをZ80PIOに接続し、  
割り込みモード2で使用する



ースIC)の出力端子をZ80PIOのポート入力に接続して、Z80の割り込みを用いて送受信するための回路を図Aに示します。

このような手法を用いれば、Z80ファミリのICとしてZ80PIOだけでも使い慣れておけば、他の使い慣れたZ80ファミリ以外のICもZ80のモード2割り込みで簡単に利用できるようになります。



## Z80PIO の使用上の注意点

Z80PIO には、普段使っているときには気が付かないような問題など、注意すべき点がいくつかあります。そのあたりをちょっと整理しておきます。

### ● 電源 OFF 時/初期化時の端子の状態の変化

PIO はパワー ON(リセット)時にはチャンネル A, B ともに入力に設定されます。ということは、外部からみればすべてハイ・インピーダンス状態になるということです。

通常はこのポートの端子にはプルアップ抵抗を接続し、これによってハイ・インピーダンス状態は H レベルになります。

ここまでは大した問題ではないのですが、つぎに CPU のプログラムが PIO の初期化を開始してモード設定をするときに、出力と設定されたポートの端子はモード設定後にはすべて L レベルを出力する点に注意します。

当然、その後に出力データとして“1”を書き込めば出力端子は H レベルになりますが、モード設定直後から“1”を書き込むまでの間は、どうしても L レベルが出力されてしまいます。

これは、短くすれば数  $\mu\text{s}$  程度の一般には短いと考えられる時間かもしれませんが、場合によっては 0.1  $\mu\text{s}$  でも L レベルが出ると困ることも多くあります。

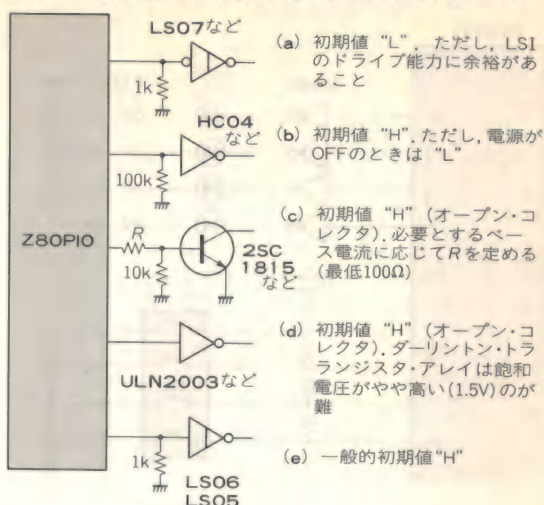
この問題には図 11 に示すような回路で対処する方法があります。これには(a)から(e)まで五つの方式を示しています。図(a)は、ポートからバッファ IC(ISO7)を接続していますが、これにプルダウン抵抗を付加することで初期値は L レベルとなります。

図(b)は、初期値 H レベルを得るためにバッファではなくインバータを入れています。しかしこのインバータの出力回路の構成から、電源が入っていないと L レベルを出力してしまうので、これではまだ問題が残ります。

図(c)は、トランジスタを用いてオープン・コレクタで出力しています。これならば初期状態で H レベルが出力され、さらに電源 OFF 時もトランジスタは OFF なので、コレクタの先に接続された回路には影響を与えません。

図(d)も同様の考え方で、ダーリントン・トランジスタ・アレイによるものです。比較的大電流のドライブ向きです。ただし一般にダーリントン・トランジスタ・アレイは LS-TTL ほど高速ではありませんし、飽和電圧がやや高い(約 1.5 V)ことなど多少注意すべき点もあります。とはいっても ULN2003/2083 などは還流用などにも使えるダイオードも内蔵していますし、リレーなどを直接駆動するならばこの方法はかな

〈図 11〉 停電時/初期化時に不必要な信号を出さないために



り有効だと思います。

最後に図(e)に最も一般的と思われる方法を示します。基本的に図(c)と同様の考え方のもので、オープン・コレクタの TTL を使っています。まとまったポート出力がある場合にはこのように TTL を使ったほうがスマートになる場合が多いと思います。

### ● モード 2 で、チャンネル B がビット・モードのとき

モード 2(バイト入出力モード)で、チャンネル A, B の割り込みルーチンで送受信をしているとき、チャンネル B でモード 3 のビット・モードの割り込みを使用することはできません。

これは、ポート B のストロブ入力による割り込みとビット・モードの割り込みとを区別できない場合があるからです。

コラム(p.98)に示すように、Z80CPU に Z80 ファミリではない LSI を接続し、割り込みで使用するなど、割り込みコントローラのない Z80 では PIO などの未使用ピンを割り込み発生源に使用するというテクニックがあります。

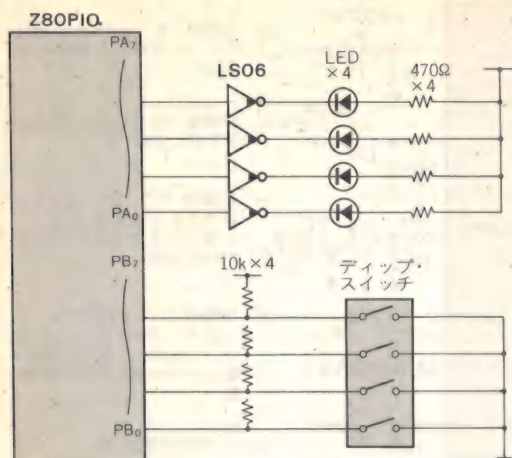
モード 2 のとき、チャンネル B をビット・モードの割り込みで使いたいときは、モード 2 の入力によるチャンネル B の割り込みと、ビット・モードでの割り込みを、きちんと区別できるような回路を設計する必要があります。

### ● モード 3(ビット・モード)の割り込みはエッジ・センス

ビット・モードの割り込みは H レベルや L レベルで発生するのではなく、L レベルから H レベルへの立ち上がりか、あるいは H レベルから L レベルへの立ち下がりで発生します。

つまり、立ち上がりエッジで割り込みするようにプログラムしてある場合は、いったん L レベルから H

〈図 12〉 ディップ・スイッチ入力と LED 出力の回路



レベルに立ち上がったときに 1 回だけ CPU に割り込みがかかります。その後、再び L レベルに戻ってから、また H レベルに立ち上がらない限り CPU に割り込みはかかりません。

### ● モード 3 (ビット・モード) の OR 条件で割り込みを使用する場合

Z80PIO をモード 3 の OR 条件で割り込みを使用する場合、ちょっと面倒な問題があります。それは、OR 条件になっているビットの中でどのビットの変化によって、割り込みが発生したのかを判定するための完全な情報を Z80PIO が返してくれないことです。

判定はポートのデータを読んでビットの状態を見れば良いのですが、例えばある二つのビットの、どちらかが H レベルになれば割り込みがかかるというようにプログラムしてある場合、どちらのビットも“1” (つまり H レベル) だったときなどです。どれが変化のあったビットなのかを確実に調べることは、割り込み先のプログラムだけでは不可能です。

このような場合、完全に判別できるように独立した割り込みベクタを発生できるように Z80 ファミリーを 1 個追加するなど、システムを変更するのが理想的といえは理想的ですが…、筆者ならこうするという方法を次に示します。

▶ Z80CTC など別の割り込みに利用できる入力端子を利用する

Z80CTC には 4 チャンネルのカウンタ入力があり、これらを用いることで 1 個の Z80CTC につき、4 入力までの割り込みを正確に発生することができます (詳しくは第 8 章を参照)。

▶ プログラムで一定周期ごとにポートの入力信号を監視する

Z80PIO の割り込みは使用せず、Z80CTC などのタイマ割り込みを一定周期で発生させ、そこで定期的に

〈リスト 1〉 図 12 の回路用のテスト・プログラム例

```

..... 基本定数 .....
: 動作環境に応じて修正
PIOAD equ 1Ch ; P I OポートAデータ
PIOAC equ PIOAD+1 ; P I OポートAコントロール
PIOBD equ PIOAD+2 ; P I OポートBデータ
PIOBC equ PIOAD+3 ; P I OポートBコントロール

..... 初期化 .....
SwLed:
..... P I OポートAの初期化
ld a, 11001111b ;モード3に設定
out (PIOAC), a
ld a, 00000000b ;全ビット出力
out (PIOAC), a
ld a, 00000111b ;割り込みは使用しない
out (PIOAC), a
..... P I OポートBの初期化
ld a, 11001111b ;モード3に設定
out (PIOBC), a
ld a, 11111111b ;全ビット入力
out (PIOBC), a
ld a, 00000111b ;割り込みは使用しない
out (PIOBC), a

..... メイン・プログラム .....
swled_loop:
..... ディップ・スイッチ (ポートB) より入力
in a, (PIOBD)
..... LED (ポートA) へ出力
out (PIOAD), a
..... 繰り返し
jr swled_loop
    
```

PIO の端子をチェックするということです。ただし、当然タイマ割り込みによるプログラム処理のオーバ・ヘッドが多少あります。また最大の問題は信号の変化から判定までの時間の遅れが、タイマ割り込みの周期の時間だけ遅れることです。

▶ チャンネル A/B とも、ビット・モードで割り込みを使用

これは両チャンネルの入力端子はともに同様の接続をして、チャンネル A は H レベル条件 (立ち上がり) で割り込みをかけ、チャンネル B は反対に L レベル条件 (立ち下がり) で割り込みをかけるようにする方法です。

これならば、Z80CTC などを使わずに済みます。しかしポートを余計に使用することや、プログラム処理のオーバ・ヘッドを考えるとあまり現実的な方法ではないでしょう。

## Z80PIO の応用例

### ● ディップ・スイッチ入力と LED 出力

まずは、4 ビットのディップ・スイッチ入力と 4 ビットの LED 出力を行う例から示します。回路例を図 12 に、プログラム例をリスト 1 に示します。

動作は単純にチャンネル B のディップ・スイッチから入力したデータをチャンネル A に出力するだけのものです。チャンネル A、B ともに下位 4 ビットだけを使用していますが、この例では上位 4 ビットは未使用なので、プログラムでは特に意識せずに 8 ビット入力してそのまま 8 ビットを出力しています。

Z80PIO はモード 3 (ビット・モード) で使います。



割り込みなどは一切使用しません。

Z80PIO の使い方としては、このビット・モードが最も多いのではないかと思います。Z80PIO の基本中の基本です。

●セントロニクス準拠プリンタ・インターフェース(出力)

Z80PIO はチャンネル A をモード 0(出力モード)で使います。チャンネル B はこの例では未使用です。割り込みはチャンネル A で使用しています。

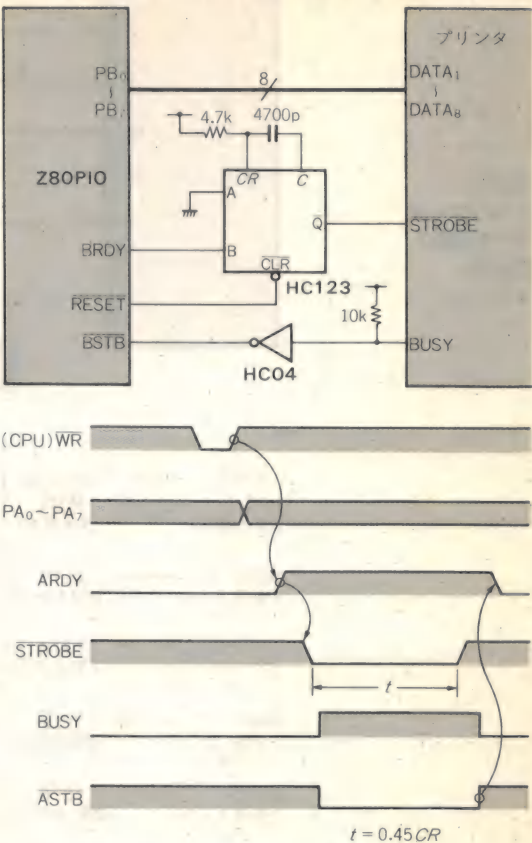
回路例を図 13 に、プログラム例をリスト 2 に示します。

CPU がデータを PIO に書き込むと、まず ARDY 端子が H レベルになります。これをトリガとしてワンショット・マルチバイブレータを使ってプリンタに書き込みクロック(ストロブ)を出力します。

プリンタ側では書き込みクロックが入力されると BUSY 端子を H レベルにし、1 文字受信処理の終了で L レベルに戻します。これが PIO の  $\overline{\text{ASTB}}$  に入力されているので、割り込みが発生し、次の印字データの出力となるわけです。

このプログラムを実行すると、プリンタに “Hello

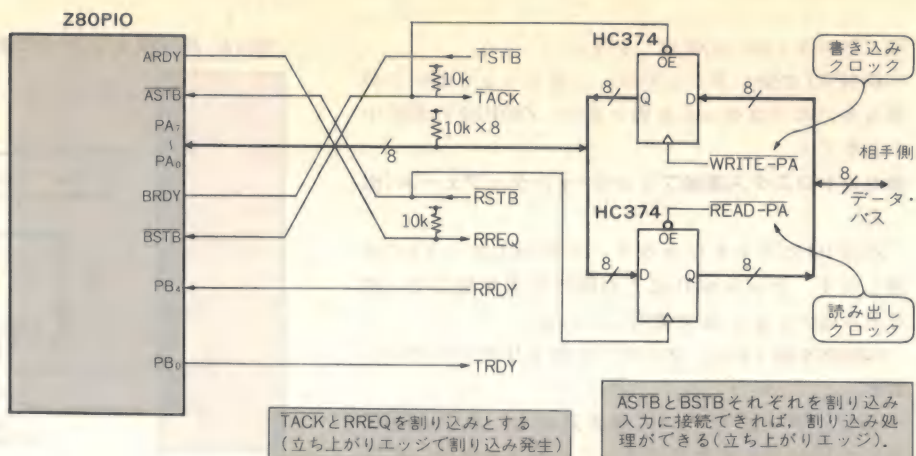
〈図 13〉 Z80PIO をモード 0 で使ったプリンタ出力回路の例



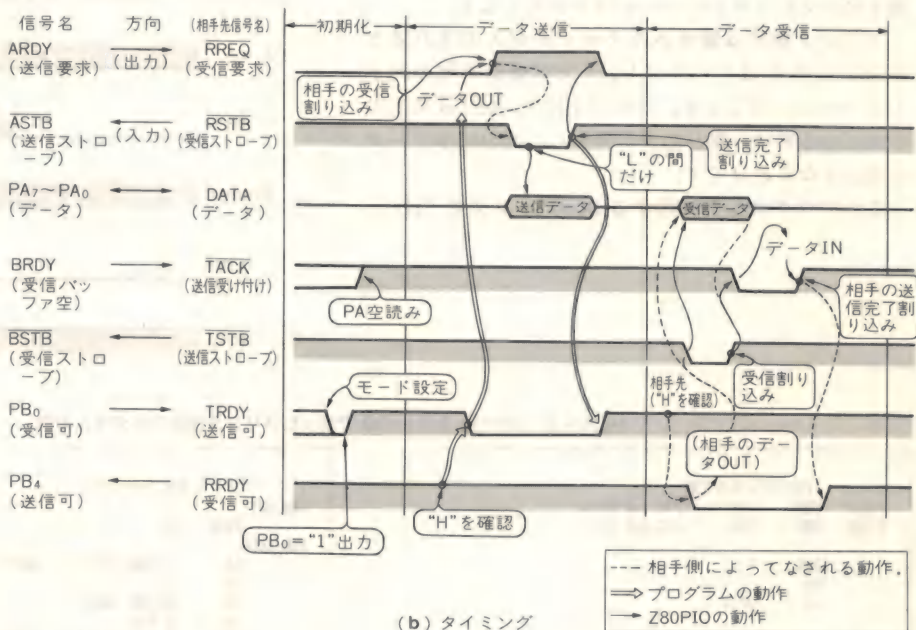
〈リスト 2〉 Z80PIO をモード 0 で使ったプリンタ出力プログラムの例

<pre>; I/Oアドレスの定義 PIOAD EQU 1Ch ;PIO Ach Data PIOAC EQU 1Dh ;PIO Ach Ctrl  ORG 0 DI LD SP,0  ; PIO初期化 DI ;割り込みモード設定 IM 2 LD A,HIGH PIO_INT LD I,A  LD A,00001111B ;PIOモード0 OUT (PIOAC),A LD A,10000111B ;割り込み使用する OUT (PIOAC),A LD A,LOW PIO_INT ;割り込みベクタ設定 OUT (PIOAC),A  XOR A ;ワークエリア初期化 LD (PRN_BUSY),A  EI LD HL,PRN_DATA LOOP: LD A,(HL) ;文字データ INC HL OR A JR Z,PRN_END CALL PRN_OUT ;プリンタ出力 JR LOOP  PRN_END: DI HALT ;CPU停止</pre>	<pre>; プリンタ出力ルーチン PRN_OUT: PUSH AF PRN_LOOP: LD A,(PRN_BUSY) ;BUSYフラグチェック OR A JR NZ,PRN_LOOP LD A,255 LD (PRN_BUSY),A ;BUSYフラグセット POP AF ;印字文字データ OUT (PIOAD),A RET  ; PIO送信割り込みルーチン PIO_ACH: PUSH AF XOR A LD (PRN_BUSY),A ;BUSYフラグクリア POP AF EI RETI  ; 割り込みベクタ・テーブル ORG (\$+1) AND 0FFEH PIO_INT: DEFW PIO_ACH  PRN_DATA: DEFB "CQI CQI",13,10,0 ;テスト印字データ  PRN_BUSY: ORG 8000H ;RAM領域 DEFS 0 ;BUSYフラグ  END</pre>
--	---

〈図 14〉  
二つの CPU 間  
データ通信の  
回路例と  
タイミング



(a) 回路例



(b) タイミング

## オリジナル+αのZ80PIO

シャープ製のCMOS版Z80PIO(LH5081L)は、オリジナルのザイログ製Z80PIOにはない、PIOのモードや $\overline{\text{STB}}$ 入力端子やRDY出力端子の状態を読み出す機能が拡張されています。オリジナルのZ80PIOでは、各チャンネルのコントロール・ポートは書き込み専用で、読み出しができませんが、シャープ製のZ80PIOではどちらのチャンネルのコントロール・ポートでも両方のチャンネルの状態を読み出せます(図B)。これを使うとモード3で使用する場合、 $\overline{\text{ASTB}}$ と $\overline{\text{BSTB}}$ の入力が2ビット増えることになります。

〈図 B〉ステータス情報ワード

D7	D6	D5	D4	D3	D2	D1	D0
AM <sub>1</sub>	AM <sub>0</sub>	ARDY	$\overline{\text{ASTB}}$	BM <sub>1</sub>	BM <sub>0</sub>	BRDY	$\overline{\text{BSTB}}$

Aポート・モード設定

Bポート・モード設定

M <sub>1</sub>	M <sub>0</sub>	モード
0	0	モード0:出力モード
0	1	モード1:入力モード
1	0	モード2:双方向モード(Aポートのみ)
1	1	モード3:ビット・モード



```

;----- 基本定数 -----
; 動作環境に応じて修正
INTVEC equ 0F000h ; スタック&割り込みベクタ開始アドレス
INTPIO equ INTVEC+00h ; PIO-A ~ 割り込みベクタ開始アドレス
PIOAD equ 1Ch ; PIOポートAデータ
PIOAC equ PIOAD+1 ; PIOポートAコントロール
PIOBD equ PIOAD+2 ; PIOポートBデータ
PIOBC equ PIOAD+3 ; PIOポートBコントロール
;----- プログラム -----
TwoCpuTest:
;----- 初期化 (スタック&割り込み設定) -----
di ; ベクタ書き換え等のため割り込み禁止
ld sp, INTVEC ; スタック・ポインタを初期化
ld a, high INTVEC ;
ld i, a
im2
;----- C P U 間通信のための初期化 -----
ld ix, INTPIO
call CmnInit
ei ; 割り込み許可が必要

;----- C P U 間通信テスト (Z80PIO側) -----
; 00h~FFhを順に送信して受信のくりかえし
tm20:
ld a, 00h
push af
call CmnOut ; 送信
call CmnIn ; 受信
pop af
inc a
jr tm20

;----- C P U 間通信制御モジュール -----
;----- C P U 間通信のための初期化 -----
CmnInit:
;----- 送信/受信の割り込み先をベクタに書き込む -----
ld de, txint
ld (ix+0), e ; PIO-A 割り込みベクタ内へ
ld (ix+1), d ; txint アドレスを書き込む
ld de, rxint
ld (ix+2), e ; PIO-B 割り込みベクタ内へ
ld (ix+3), d ; rxint アドレスを書き込む
;----- P I O ポート A の初期化 -----
ld a, 10001111b ; モード 2 に設定
out (PIOAC), a
ld a, 10000111b ; 割り込みを使用 (ポート A : 送信)
out (PIOAC), a
push ix
pop de ; de = ix
ld a, e
out (PIOAC), a ; ポート A 割り込みベクタ
;----- P I O ポート B の初期化 -----
ld a, 11001111b ; モード 3 に設定
out (PIOBC), a
ld a, 11110000b ; ビット 7~4: 入力, ビット 3~0: 出力
out (PIOBC), a
ld a, 10110111b ; 割り込みを使用 (ポート B : 受信)
out (PIOBC), a
ld a, 11111111b ; ビット割り込みは使用しない(できない)
out (PIOBC), a
inc de
inc de
ld a, e
out (PIOBC), a ; ポート B 割り込みベクタ
;-----
in a, (PIOAD) ; BRDY=H にするために

```

```

xor a ; 一度データを空読みする
ld (rxflg), a ; 受信未処理
ld a, 00000001b ; フラグ = 0 (処理済)
out (PIOBD), a ; PB0=1 (受信可)
ret

```

```

;----- 1 バイト送信 -----
CmnOut:
ld b, a ; b = データ
;----- 送信完了 (PB0=1) かつ送信可 (PB4=1) まで待つ -----
in a, (PIOBD)
bit 0, a ; PB0=1 なら送信完了
jr z, c10
bit 4, a ; PB4=1 なら送信可
jr z, c10
;----- データ送信 -----
res 0, a ; PB0=0 (受信不可)
out (PIOBD), a
ld a, b ; a = データ
out (PIOAD), a
ret

```

```

;----- 1 バイト受信 -----
CmnIn:
;----- 受信未処理フラグ (rxflg) がセットされるまで待つ -----
; (受信割り込み待ち)
ld hl, rxflg
xor a ; a=0
or (hl)
jr z, c10
;----- 受信データ (rxdata) 取得して rxflg=0 にクリア -----
ld (hl), a ; rxflg=0
ld a, (rxdata) ; a = データ
ret

```

```

;----- 送信完了割り込みルーチン -----
txint:
push af
in a, (PIOBD)
set 0, a ; PB0=0 (受信可)
out (PIOBD), a
pop af
reti

```

```

;----- 受信割り込みルーチン -----
rxint:
push af
ld a, 1 ; rxflg=1
ld (rxflg), a
;
in a, (PIOAD) ; データ受信
ld (rxdata), a ; rxdata = 受信データ
pop af
reti

```

```

org 8000h
;----- モジュール用変数 -----
rxflg: defs 1 ; 受信未処理フラグ (0:処理済)
rxdata: defs 1 ; 受信データ

```

Printer!”と1行だけ印字します。このプログラムは最初の1文字目の出力以降は、必要なときだけPIO割り込みで出力するようになっているので、印刷処理に費やされる時間の無駄が最小限に抑えられています。

またリストをみるとわかりますが、PIOの初期化ルーチンPrnInitを呼び出すときに、IXレジスタに割り込みベクタ・テーブルのアドレスを入れて呼び出しています。そしてRAM領域に割り込みベクタを設定しています。これも初期化のテクニックの一つです。

## ● 二つのCPU間でのデータ通信

Z80PIOはモード2(双方向モード)で使います。割り込みは出力にチャンネルA、入力にチャンネルBと両方のチャンネルを使用します。

ブロック図を図14に、プログラム例をリスト3に示します。

一般的には片側だけZ80CPUで、もう片側はまったく別のCPUやDSPなどの場合が多いと思います。そのような場合でも、8ビットのデータ・バスと書き込みクロックと読み出しクロックが取れば、ほとんどのCPUでインターフェースがとれるでしょう。当然ハンドシェイクのタイミングなどは注意してプログラムする必要があります。

ここにあげた回路は、いずれも距離が十数cm程度のごく短い間の通信用です。数十cmのケーブルなどを用いて距離をのばしたい場合は、バッファを付加するなどの対処は必要です。

## ● 参考文献

- (1) VOLUME 1 DATABOOK, MICROPROCESSORS AND PERIPHERALS, ZILOG.
- (2) Z80 Family User's Manual, ZILOG.

時間を計ったりクロック数を数える

## Z80CTCの使い方

野口智樹

### Z80CTC の概要

#### ● Z80CTC とは

Z80CTC(Counter Timer Circuit)は4個の8ビット・カウンタからなるLSIです。システム・クロックを基準にして一定時間ごとに割り込みをかけたり、外部からのクロック数をカウントしたりできます。

Z80CTC は、次のような特徴があります。

- ・4個の各チャンネルはそれぞれ独立してカウンタまたはタイマ・モードで動作できる
- ・各チャンネルはシステム・クロックを1/16または1/256するプリスケアラを内蔵(タイマ・モードでのみ有効)
- ・Z80CPUのモード2割り込みに対応(割り込み優先順位はチャンネル0が最高、チャンネル3が最低)
- ・各チャンネルのゼロ/タイム・アウト出力は、ダーリントン・トランジスタをドライブ可能
- ・カウンタ・モードまたはタイマ・モードいずれの場合にも、CPUからカウンタの内容を読み出すことが可能
- ・タイマ・モード時のタイマ動作の起動用トリガの極性をプログラム可能
- ・カウンタ・モード時のカウント動作のトリガの極性をプログラム可能

#### ● 内部ブロック図、ピン配置

Z80CTCのブロック図を図1に、ピン配置を図2に、図3と表1に電気的特性やタイミングを示します。

CPUとの接続についてはすでに説明しているので省略します(第4章参照)。

外部との入出力に使われる端子は、外部クロックまたはトリガ入力であるCLK/TRG<sub>0</sub>~CLK/TRG<sub>3</sub>と、カウント終了出力であるZC/TO<sub>0</sub>~ZC/TO<sub>2</sub>となります。

ピン配置をみるとわかりますが、Z80CTCのチャンネル3には、ZC/TO出力がありません。これはピン数の制限によるものと思われます。また44ピンであるはずのQFPについても、チャンネル3のZC/TO出力はありません。

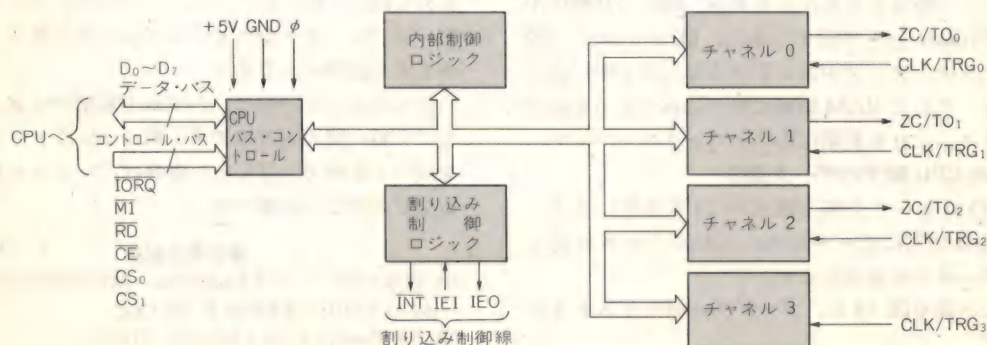
### Z80CTC の各種モード

Z80CTCには大きく分けて、システム・クロックを基準クロックとして動作するタイマ・モードと、外部クロックを基準に動作するカウンタ・モードの二つがあります。

#### ● Z80CTC カウンタ・モードの動作

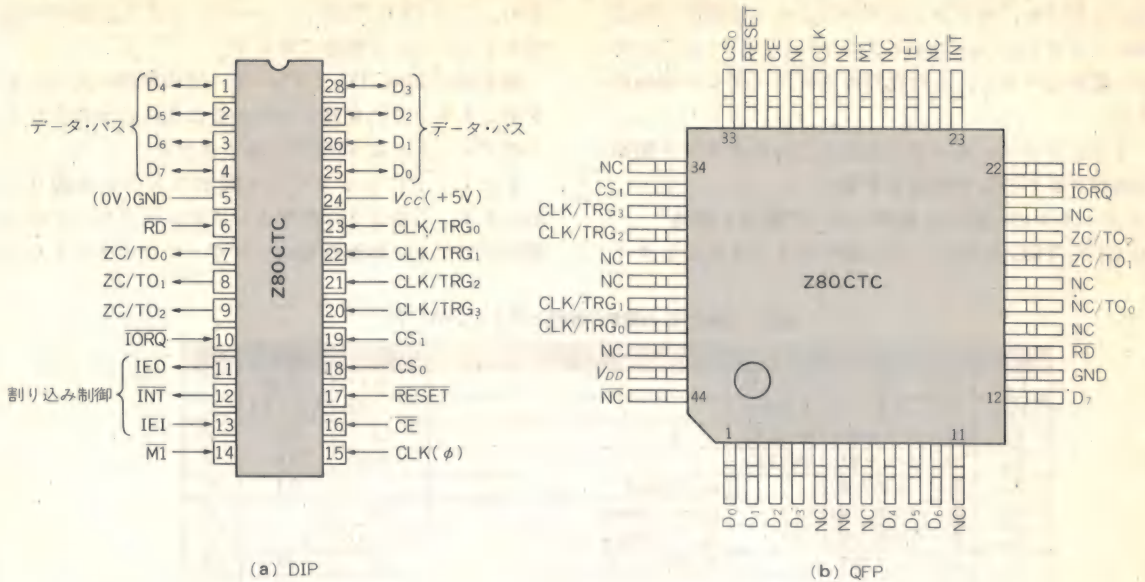
カウンタ・モードでは、CTCはCLK/TRG入力信号のエッジによりダウン・カウント動作をします。エ

〈図1〉 Z80CTC の構成





〈図2〉 Z80CTC のピン配置



ツジの極性(立ち上がりか立ち下がり)はプログラムで指定できます。

Z80CTC は、プログラムで指定された極性のエッジを検出すると、カウント値を-1 します(ダウン・カウント)。

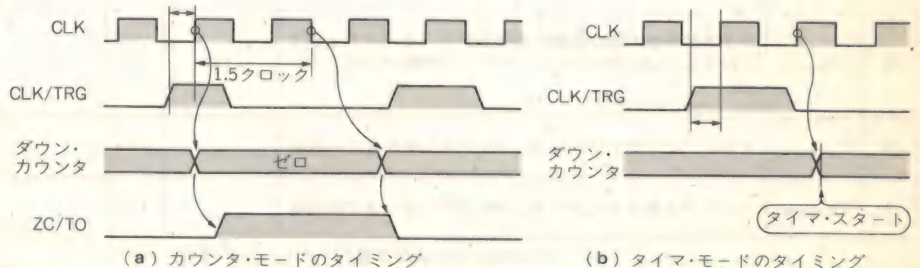
ここでカウント値が0 になると、該当するチャネルの ZC/TO 出力ピンが1.5 システム・クロックの間だけ H レベルになります。このとき、CPU にゼロ・カウントの意味をもった割り込みをかけることができます。

なお、カウント可能な外部クロックは、Z80CTC 内部がハードウェア的に8 ビットのカウンタなので、2 の8 乗、つまり256 までです。これを越えるカウントをしたい場合には、ゼロ・カウント割り込みなどを用いてソフト的に桁上げしたり、二つのカウンタをカスケード接続します。

また外部クロック周波数は、システム・クロック周波数の1/2 以下でないと、正常にカウントできません。

図4(a)に Z80CTC カウンタ・モードの動作タイミングを示します。

〈図4〉 Z80CTC の動作のようす



### ● Z80CTC タイマ・モードの動作

タイマ・モードでは、システム・クロックをまずプリスケアラに入力し分周します。そしてこの出力をカウント・ダウンします。プリスケアラは1/16 または1/256 を選択できるので、システム・クロックを $\phi$ とすると、1/16 の場合は $16\phi \sim 4096\phi$ で、1/256 の場合は $256\phi \sim 65536\phi$ の範囲でカウント可能です。

またカウントがゼロになった時点で割り込みをかけることができ、カウンタがゼロになると自動的に時定数レジスタ(タイム・コンスタント・レジスタ)からカウント値をロードし、ふたたびカウント・ダウンを始めます。

さらにタイマ・モードには次の二つのカウント開始方法がプログラムで指定できます。

- (1) チャネルの初期化と同時にタイマ動作を開始
- (2) CLK/TRG 入力にトリガ信号が入力されるとタイ

マ動作を開始

また、この CLK/TRG ピンへのトリガ入力信号の極性もプログラムで指定できます。

図 4 (b)に Z80CTC タイマ・モードの動作タイミングを示します。これを見るとわかるように、出力クロックのデューティは 50 %ではありません。

またトリガによるカウント開始のようすを図 5 に示します。このように単なるシステム・クロックの分周だけでなく、外部からのトリガ・クロックによりダ

＜表 1＞ Z80CTC の電氣的特性とタイミング

記号	項 目	測定条件	最小	最大	単位
$V_{ILC}$	クロック低レベル入力電圧		-0.3	+0.45	V
$V_{IHC}$	クロック高レベル入力電圧		$V_{CC}-0.6$	$V_{CC}+0.3$	V
$V_{IL}$	低レベル入力電圧(除くCLK)		2.2	$V_{CC}$	V
$V_{IH}$	高レベル入力電圧(除くCLK)		-0.3	0.8	V
$V_{OL}$	低レベル出力電圧	$I_{OL}=2.0\text{mA}$		0.4	V
$V_{OH1}$	高レベル出力電圧(I)	$I_{OH}=-1.6\text{mA}$	2.4		V
$V_{OH2}$	高レベル出力電圧(II)	$I_{OH}=-250\mu\text{A}$	$V_{CC}-0.8$		V
$I_{L1}$	入力リーク電流	$V_{SS}\leq V_{IN}\leq V_{CC}$	-10	10	$\mu\text{A}$
$I_{L0}$	フローティング時 3 ステート出力電流	$V_{SS}+0.4\leq V_{OUT}\leq V_{CC}$	-10	10	$\mu\text{A}$
$I_{OHD}$	ゲーリントン駆動電流	$V_{OH}=1.5\text{V}$ , $R_{EXT}=1.1\text{k}\Omega$ ZC/TO <sub>0</sub> ~ZC/TO <sub>2</sub> に適用	-1.5	-5.0	mA
$I_{CC1}$	電源電流(動作時)	$V_{CC}=5\text{V}$ $f_{CLK}=1/T_{CC}$ $V_{IH}=V_{IHC}$ $=V_{CC}-0.2\text{V}$ $V_{IL}=V_{ILC}=0.2$	6MHz 版	8	mA
			8MHz 版	10	
			10MHz 版	12	
$I_{CC2}$	電源電流(静止時)	$V_{CC}=5\text{V}$ , $\text{CLK}=V_{CC}$ $V_{IL}=V_{ILC}=0.2\text{V}$		10	$\mu\text{A}$

(a) Z80CTC の電氣的特性

番号	記号	項 目	6MHz 版		8MHz 版		10MHz 版	
			最小	最大	最小	最大	最小	最大
19	$T_{dC(INT)}$	クロック立ち上がりから $\overline{\text{INT}}$ 立ち下がりまでの遅延	$T_{CC}+120$		$T_{CC}+100$		$T_{CC}+80$	
20	$T_{dCLK}$	CLK/TRG 立ち上がりから $\overline{\text{INT}}$ 立ち下がりまでの遅延(カウンタ・モード)	$T_{sCTR(C)}$ を満足する場合		$T_{CC}+100$		$T_{CC}+80$	
			$T_{sCTR(C)}$ を満足しない場合		$1+T_{CC}+100$		$1+T_{CC}+80$	
21	$T_{eCTR}$	CLK/TRG 周期	$2T_{CC}$		$2T_{CC}$		$2T_{CC}$	
22	$T_{rCTR}$	CLK/TRG 立ち上がり時間		40		30		30
23	$T_{fCTR}$	CLK/TRG 立ち下がり時間		40		30		30
24	$T_{wCTRl}$	CLK/TRG 低レベル・パルス幅	120		90		90	
25	$T_{wCTRh}$	CLK/TRG 高レベル・パルス幅	120		90		90	
26	$T_{sCTR(CS)}$	即時カウントに要するクロックの立ち上がりに対する CLK/TRG のセットアップ時間(カウンタ・モード)	150		110		90	
27	$T_{sCTR(CL)}$	プリスケアラの即時起動に要するクロックの立ち上がりに対する CLK/TRG のセットアップ時間(タイマ・モード)	150		110		90	
28	$T_{dC(ZC/TO)}$	クロック立ち上がりから ZC/TO 立ち上がりまでの遅延		140		100		80
29	$T_{dC(ZC/TO)}$	クロック立ち下がりから ZC/TO 立ち下がりまでの遅延		140		100		180

(b) Z80CTC の入出力端子のタイミング(単位: ns)



ウン・カウントを開始することもできます。

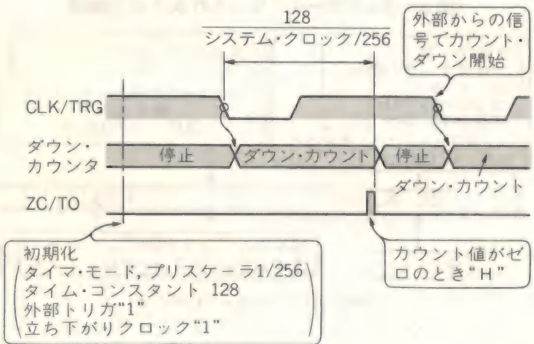
## Z80CTC のプログラミング

### ● Z80CTC へのアクセス

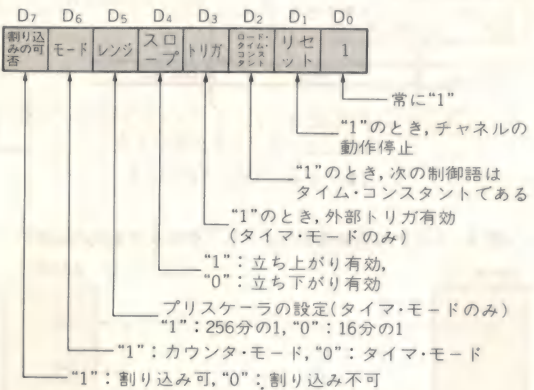
Z80CTC は CS<sub>0</sub> と CS<sub>1</sub> の 2 本のチャネル・セレクト端子があり、それぞれ表 2 のようにチャネル 0～チャネル 3 までを選択します。

チャネルを選択し書き込み動作をすると、そのチャネルのコントロール・ワードを設定することになります。また読み出し動作をすると、そのチャネルのカウント値を読み出します。

〈図 5〉 外部トリガを使ったタイマ・モード



〈図 6〉 Z80CTC のコントロール・ワード



D <sub>2</sub> ロード・タイム・ コンスタント	D <sub>1</sub> リセット	動 作
0	0	現在の動作を継続
0	1	現在の動作を停止
1	0	ダウン・カウンタの内容が 0 になった後、新しい時間定数がロードされる
1	1	現在の動作が停止し、ただちに新しい時間定数がロードされる

(a) チャネル・コントロール・ワード

### ● Z80CTC のモード設定のプログラミング方法

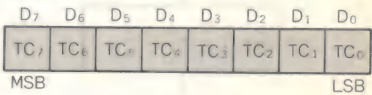
図 6 にコントロール・ワードのフォーマットを示します。タイム・コンスタントとは、カウント・ダウンを開始するときの最初の値です。

図 7 に Z80CTC の初期化プログラミングの流れを示します。これは、Z80CTC のモード設定の一般的な流れなので、これにしたがったモード設定を行えば問題になることはないでしょう。

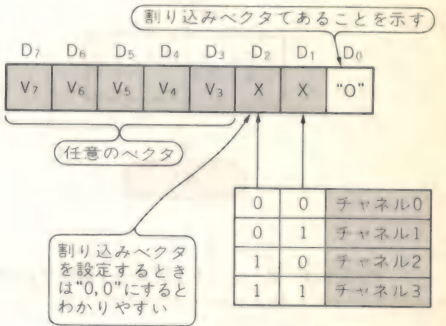
最初にいきなり割り込みベクタを書き込む点を不思議に思う方もいるでしょう。Z80 の割り込みベクタは偶数番地を指定する必要があるため、必ず最下位ビットが "0" になります。CTC に対してのコントロー

〈表 2〉 Z80CTC の各ポートの選択

CS <sub>1</sub>	CS <sub>0</sub>	セレクトされるチャネル	CPU の動作	CTC の動作
"L"	"L"	チャネル 0	リード	カウント値読み出し
			ライト	コントロール・ワード書き込み
"L"	"H"	チャネル 1	リード	カウント値読み出し
			ライト	コントロール・ワード書き込み
"H"	"L"	チャネル 2	リード	カウント値読み出し
			ライト	コントロール・ワード書き込み
"H"	"H"	チャネル 3	リード	カウント値読み出し
			ライト	コントロール・ワード書き込み



(b) タイム・コンスタント・ワード



(c) 割り込みベクタのフォーマット (チャネル 0 のみ)

ル・ワードは最下位ビットを“1”にするので、区別が  
つくわけです。

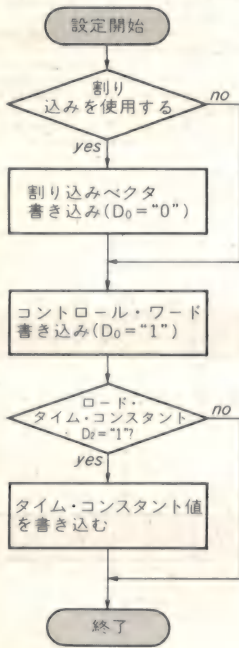
ここでコントロール・ワードのリセット(D<sub>1</sub>)について説明します。これはCTCのカウント動作を停止させるか、継続させるかの設定です。これをセットしていると、システム・クロックが入力されても外部からクロックが入力されても、カウント動作は止まったままになります。

またロード・タイム・コンスタント(D<sub>2</sub>)ビットとの組み合わせでは、すでに設定してあるダウン・カウンタが0になってから、新しく設定したタイム・コンスタントがロードされ、カウントを始めるようになります(図8)。

### ● 割り込みベクタの設定

Z80CTCは割り込みベクタの設定をチャンネル0に対してのみ行うことができます。チャンネル0以外の割り込みベクタは、チャンネル0に書かれたベクタのうち図

〈図7〉 Z80CTCの初期化の流れ



〈リスト1〉 ボーレート・クロック設定プログラム

```

..... 基本定数 .....
: 動作環境に応じて修正
CTC2 equ 12h          ; CTC Ch.2
BPSClk equ 4          ; 9600bps (CLK=9.8304MHz, SIO=X16)
: ..... ボーレート設定(CTC初期化) .....
ld a, 00000111b       ; 割り込みなし、タイム・モード、1/16
out (CTC2), a
ld a, BPSClk           ; タイム・コンスタント
out (CTC2), a

```

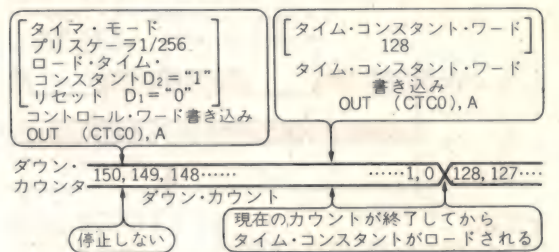
6(c)に示すように、ビット2とビット1がチャンネルごとに自動的に設定されます。

ただし、チャンネル0に割り込みベクタとして設定された値に+2されたアドレスがチャンネル1、+4されたアドレスがチャンネル2...と考えるのは誤りです。

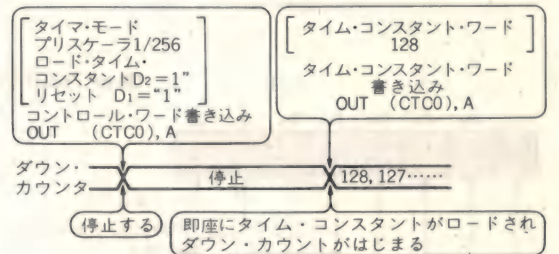
CTC内部では、ビット1とビット2にどんな値が入っていようとも、チャンネル0なら“00”で、チャンネル2なら“10”と置き換えてベクタを出力します。

よってCTCの割り込みベクタ・テーブル・アドレスとしては、アドレスの下位3ビットが“0”になるアドレスを先頭にして、チャンネル0から3までを順番に並べなければなりません。

〈図8〉 コントロール・ワードのD<sub>2</sub>とD<sub>1</sub>の関係

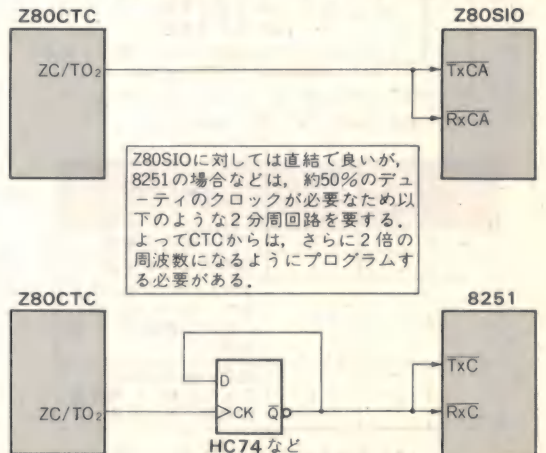


(a) リセットD<sub>2</sub>="0"のとき



(b) リセットD<sub>2</sub>="1"のとき

〈図9〉 シリアル通信用ボーレート・クロック生成の回路例





## Z80CTC の応用例

### ● シリアル通信ボーレート・クロック生成

これは Z80CTC としてはもっとも使用頻度が高い用途ではないでしょうか。回路例が図 9 で、プログラム例はリスト 1 です。

すでに説明したようにタイマ・モードであっても Z80CTC の出力はデューティ 50 % ではないので、8251 などの送受信クロックとしてはそのままでは使えません。50 % のデューティにするために、フリップフロップを一つ追加して 2 分周してから 8251 に接続します。もちろんこの場合には、Z80CTC の出力周波数を 2 倍にして考える必要があります。

Z80SIO を使う場合はそのまま ZC/TO 出力を送受信クロック端子に接続していただいじょうぶです。

CTC の分周値の計算式は次式となります。

システム・クロック ÷ (ボーレート × SIO 倍率) ÷ 16  
例えばシステム・クロック 9.8304 MHz, プリスケアラ 1/16, 通信速度 9600 bps, SIO のクロック倍率 × 16 とした場合,

$$9830400 \div (9600 \times 16) \div 16 = 4$$

つまり、9.8304 MHz ÷ 16 ÷ 4 = 153600 Hz のクロックが出力されます。

### ● インターバル・タイマ

この例も CTC の代表的な使用例といえるものです。インターバル・タイマとは、一定周期ごとにかかるタイマ割り込みのことです。タイマ割り込みそのものにハードウェアはとくに必要ありません。リスト 2 にプログラム例を示します。

例えば 1/60 秒ごとにポートをチェックしたいなどのときに応用します。リスト中のタイマ割り込みルーチンに処理を埋め込めばよいだけです。

### ● 回転数測定

Z80CTC は、ハードウェア的には 8 ビットのカウン

## 〈リスト 2〉 インターバル・タイマのプログラム例

```

;----- 基本定数 -----
;----- 動作環境に応じて修正 -----
INTVEC equ 0F000h ;スタック&割り込みベクタ開始アドレス
INTCTC equ INTVEC+10h ;CTC Ch0 ~ 割り込みベクタ開始アドレス
CTCO equ 10h ;CTC Ch.0
TINTCYC equ 64 ;タイマ割り込み周期の定数
;----- プログラム -----
cseg
;----- 初期化 (スタック&割り込み設定) -----
di ;ベクタ書き換え等のため割り込み禁止
ld sp, INTVEC ;スタック・ポインタを初期化
ld a, high INTVEC
ld i, a
im2
;----- インターバル・タイマのための初期化 -----
ld ix, INTCTC
ld iy, intsv
ld c, TINTCYC
call TimInit
ei ;割り込み許可すること
;+++++ 動作テスト ++++++
; (メイン処理は、intsv: で行われる)
jr $ ;何もせず、単にくりかえし
;+++++ インターバル・タイマ割り込みルーチン ++++++
; 割り込み毎にスイッチを一時ONしてOFF
intsv: push af
; ここに何らかのタイマ割り込みルーチンを書き込む
pop af
ei
reti
;----- インターバル・タイマ・モジュール -----
;----- インターバル・タイマのための初期化 -----
TimInit:
;----- CTC0 設定 -----
push ix
pop de ;de = ix
ld a, e
out (CTCO), a ;CTC 割り込みベクタ
push iy
pop de ;de = iy
ld (ix+0), e ;CTC Ch0 割り込みベクタ内へ
ld (ix+1), d ;timint アドレスを書き込む
ld a, 10100111b ;割り込みあり タイマ・モード 1/256
out (CTCO), a
ld a, c ;タイマ・コンスタント
out (CTCO), a
;-----
ret

```

タですが、最近では 16 ビットのカウンタがマイコン用カウンタ IC としては主流のようです。とはいっても、8 ビットで困ることはほとんどありません。それは、プログラムで適当に桁上げてあげれば簡単に 16 ビットでも 32 ビットでも拡張できるからです。

## Z80CTC を割り込みコントローラとして使う

Z80PIO の章でも少し触れましたが、Z80CTC の未使用カウンタを使って、Z80 以外の LSI を Z80 のモード 2 の割り込みで使うことができます。

Z80CTC には 4 チャネルのカウンタ入力があるので、これらを用いることで 1 個の Z80CTC につき 4 入力までの割り込みを発生させることができます。

Z80CTC はカウンタ・モードに設定し、タイム・コンスタント・レジスタに 1 を設定しておけば、割り込みエッジ入力後にカウント値が 0 になって、CPU

に対して割り込みが発生します。

また都合の良いことに、Z80PIO と同じように割り込み入力信号のエッジをプログラムで指定することができます。

さらに二つ以上の割り込み入力信号が同時にいった場合なども、Z80CTC の各チャネルは独立して割り込みベクタを出力するので問題ありません。このような場合も、割り込みの優先順位はチャネル 0 が最優先となります。

# 〈リスト 3〉 回転数測定プログラムの例

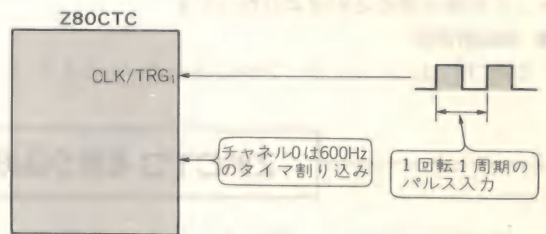
<pre> ;----- 基本定数 ----- ; 動作環境に応じて修正 INTVEC equ 0F000h ; スタック&amp;割り込みベクタ開始アドレス INTCTC equ INTVEC+10h ; CTC Ch0 ~ 割り込みベクタ開始アドレス CTC0 equ 10h ; C T C Ch. 0 CTC1 equ CTC0+1 ; C T C Ch. 1 T600HZ equ 64 ; 600Hz周期タイマ割り込みの定数 ; (RpsInit: の説明を参照)  ;----- プログラム ----- RpsTest: ;..... 初期化 (スタック&amp;割り込み設定) di ; ベクタ書き換え等のため割り込み禁止 ld sp, INTVEC ; スタック・ポインタを初期化 ld a, high INTVEC ; ld i, a ld im2 ;..... 回転数測定のための初期化 ld ix, INTCTC ld c, T600HZ call RpsInit ei ;..... 現在の回転数(rps)を表示 tstl0: call RpsGet ; 回転数が hl に格納 call DispHL ; hl を表示 jr tstl0 ; くりかえし  ;===== 回転数測定モジュール ===== ;..... モジュール用定数 ..... mtime equ 600 ; 測定周期 [600=1秒:rps] ;----- 回転数測定のための初期化 ----- RpsInit: ;..... 変数初期化 ld hl, mtime ld (rrem), hl ld hl, 0 ld (rcnt), hl ld (rps), hl ;..... CTC0 設定 push ix pop de ; de = ix ld a, c ld (CTC0), a ; CTC 割り込みベクタ・ワード ld de, timint ld (ix+0), e ; CTC Ch0 割り込みベクタ内へ ld (ix+1), d ; timint アドレスを書き込む ld a, 10100111b ; 割り込み有、タイマモード、1/256 ld (CTC0), a ld a, c ; タイムコンスタント ld (CTC0), a ;..... CTC1 設定 ld a, 01000111b ; 割り込み無、カウンタモード、立ち下がり ld (CTC1), a ld a, 0 ; タイムコンスタント ld (CTC1), a ;..... ret </pre>	<pre> ;----- 回転数測定 (タイマ割り込み) ----- timint: push af push bc push de push hl ;..... ld de, (rrem) ld a, d or e ; rrem=0 ならば jr z, ti50 ; 回転数を決定 dec de ; rrem ← rrem-1 ld (rrem), de ;..... カウント ld hl, (rcnt) in a, (CTC1) ld b, a ld a, (prvnt) ; 1つ前のカウント←現在のカウン sub b ; の結果を a に代入する ld e, a ld d, 0 add hl, de ; カウント累計 (rcnt) に16bit加算 ld (rcnt), hl ld a, b ld (prvnt), a ; 1つ前のカウント←現在のカウン jr ti90  ti50: ;..... rrem=0 になったので回転数を決定 ld hl, (rcnt) ld (rps), hl ; rcnt値がそのままrps値となる ld hl, 0 ld (rcnt), hl ld hl, mtime ; 次の回の割り込みのために ld (rrem), hl ; rremを再設定  ti90: ;..... pop hl pop de pop bc pop af ei reti  ;----- 回転数測定値取得 ----- RpsGet: ld hl, (rps) ; 現在の決定値をそのまま返す ret  ;..... org 8000 ; RAM 領域 ;..... モジュール用変数 ..... rrem: defs 2 ; 最終値決定までの割り込み回数残り rcnt: defs 2 ; 現在までの回転カウンタ累計 rps: defs 2 ; 測定回転数 (最終値) prvnt: defs 1 ; 1つ前の C T C カウント値 </pre>
--	--

回路例を図 10 に、プログラム例をリスト 3 に示します。

このプログラムでは、チャンネル 0 は一定周期のタイマ割り込みとして用い、チャンネル 1 を回転クロック・カウンタとして用いています。

ポイントは、チャンネル 1 のカウント値をチャンネル 0 のタイマ割り込みごとに、16 ビットのカウンタ値に桁上げしていることです。こうすることで一定周期ごとに正確なカウンタ値を取り込むことができます。周期は 600 Hz (0.001666..秒) とプログラムしていますが、この間に 256 以上のカウンタ値が入力されると誤動作してしまうので、計測できる最高周波数が決まります。

〈図 10〉 回転数測定の回路例



## 参考文献

- (1) VOLUME 1 DATABOOK, MICROPROCESSORS AND PERIPHERALS, ZILOG.
- (2) Z80 Family User's Manual, ZILOG.



# Z80SIOの使い方

高見 豊

## Z80SIO の概要

### ● Z80SIO とは

Z80SIO (Serial Input Output) は Z80 ファミリのシリアル通信コントローラ LSI で、1 パッケージに独立した 2 チャンネルの USART (Universal Synchronous Asynchronous Receiver and Transmitter: 汎用同期非同期送受信器) を内蔵しています。また、SIO は非同期(調歩同期)から、BSC などのキャラクタ同期、HDLC などのフラグ同期まで対応できます。さらに送受信割り込みはもちろん、DCD や CTS などのモデム・ステータスの変化によっても割り込みが可能のため、ポーリングによるソフトウェアの負担を最小限に

抑えることができます。

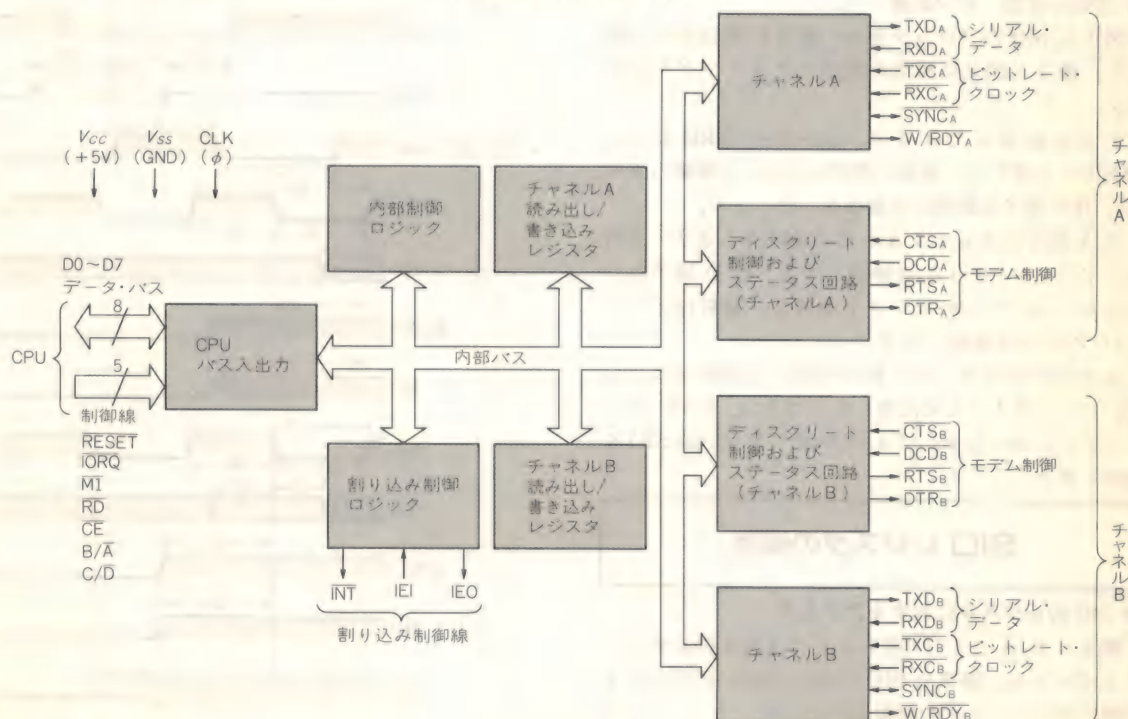
SIO は、非同期通信で 1.2 Mbps (クロック 6 MHz の場合) の高速通信が実現できます。

また SIO と同じザイログ社の汎用 (CPU を選ばない) USART である SCC は SIO と上位互換性があるため、SIO の使い方を知っていれば SCC も難しくはありません。

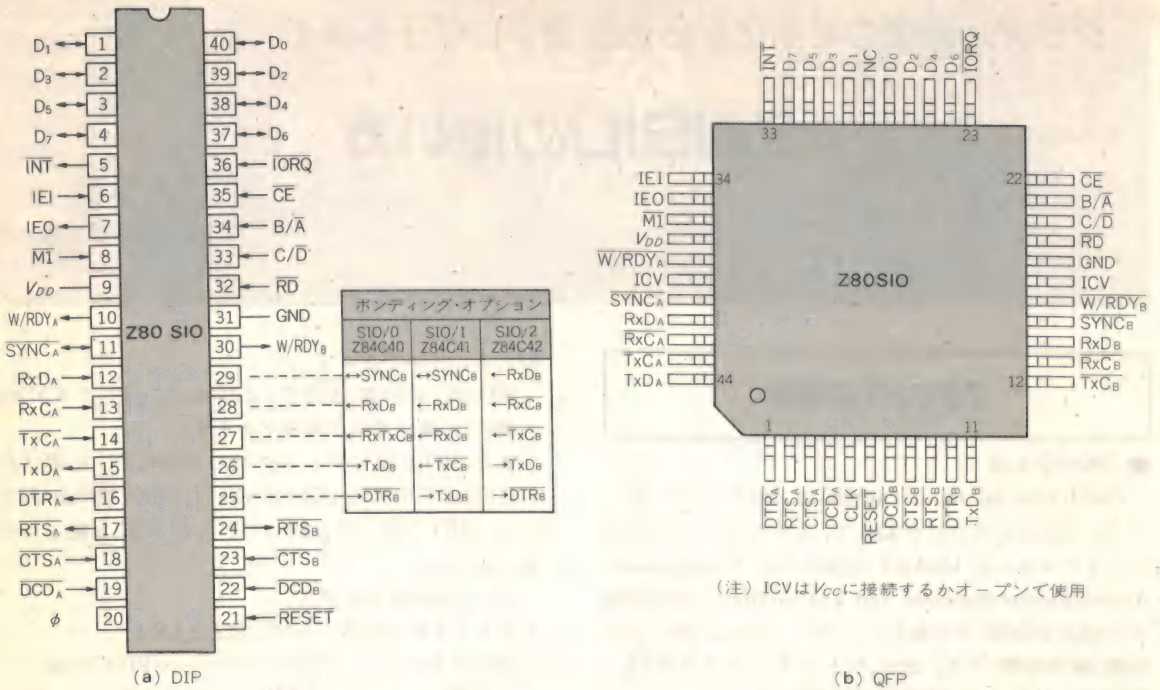
以下に特徴を示します。

- 2 チャンネルの独立した全二重 USART
- 受信は 4 バイト、送信は 2 バイトの FIFO 内蔵
- 1.2 Mbps (クロック 6 MHz) までの転送レート対応
- Z80 ファミリのデジィ・チェーン割り込みをサポート
- 非同期、キャラクタ同期 (BSC)、フラグ同期

〈図 1〉 Z80SIO のブロック図



〈図2〉 Z80SIO のピン配置



(HDLC)が可能

- ・CRC生成, チェック機能内蔵
- ・HDLC/SDLC コンパチブル
- ・CCITT-X.25 コンパチブル

#### ● ブロック図, ピン配置

図1にSIOのブロック図を, 図2にSIOのピン配置を, 表1と図3に電気的特性とタイミングを示します。

すでに説明したように, D<sub>0</sub>~D<sub>7</sub>,  $\overline{\text{IORQ}}$  など, Z80CPUと同じピン名称の端子はCPUと接続します。IEI, IEO端子も説明の必要はないでしょう。

B/ $\overline{\text{A}}$ 端子はチャンネルAとBの選択をします。通常はアドレス・バスのA<sub>1</sub>を接続します。C/ $\overline{\text{D}}$ 端子はコントロール/データ・セレクト信号で, 通常はアドレス・バスのA<sub>0</sub>を接続します。

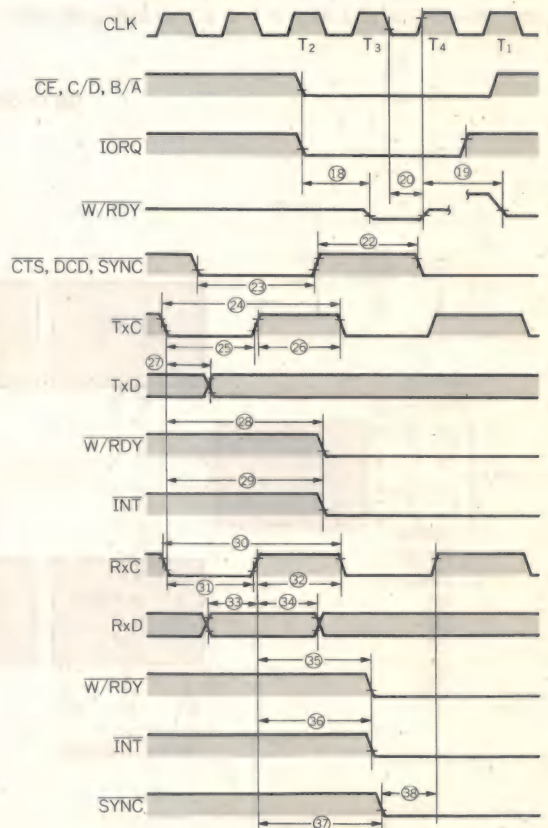
またDIP版では, ピン数の制限から接続されていないピンがあることに注意してください。目的とするシステムに合ったボンディング・オプションのSIOを選択します。

### SIOレジスタの概要

#### ● SIOのレジスタに対するアクセス

表2にB/ $\overline{\text{A}}$ とC/ $\overline{\text{D}}$ 端子により選択されるチャンネルとポートを, 表3にSIOの書き込みレジスタおよび読み出しレジスタの概要を示します。

〈図3〉 Z80SIOのタイミング図





〈表 1〉 Z80SIO の電気的特性とタイミング

記 号	項 目	測 定 条 件	最 小	最 大	単位
$V_{ILC}$	クロック低レベル入力電圧		-0.3	$\pm 0.45$	V
$V_{IHC}$	クロック高レベル入力電圧		$V_{CC}-0.6$	$V_{CC}+0.3$	V
$V_{IL}$	低レベル入力電圧(除く CLK)		2.2	$V_{CC}$	V
$V_{IH}$	高レベル入力電圧(除く CLK)		-0.3	0.8	V
$V_{OL}$	低レベル出力電圧	$I_{OL}=2.0\text{ mA}$		0.4	V
$V_{OH1}$	高レベル出力電圧(I)	$I_{OH}=1.6\text{ mA}$	2.4		V
$V_{OH2}$	高レベル出力電圧(II)	$I_{OH}=-250\text{ }\mu\text{A}$	$V_{CC}-0.8$		V
$I_{LI}$	入力リーク電流	$V_{SS}\leq V_{IN}\leq V_{CC}$	-10	10	$\mu\text{A}$
$I_{LO}$	出力リーク電流	$V_{SS}+0.4\leq V_{OUT}\leq V_{CC}$	-10	10	$\mu\text{A}$
$I_{L(SY)}$	SYNC 端子入力電流	$V_{SS}+0.4\leq V_{IN}\leq V_{CC}$	-40	10	$\mu\text{A}$
$I_{CC1}$	電源電流(動作時)	$V_{CC}=5\text{ V}$ $f_{CLK}=1/T_{CC}$ $V_{IHC}=V_{IH}$ $=V_{CC}-0.2\text{ V}$ $V_{ILC}=V_{IL}$ $=0.2\text{ V}$	6 MHz 版	10	mA
			8 MHz 版	12	
			10 MHz 版	15	
$I_{CC2}$	電源電流(静止時)	$V_{CC}=5\text{ V}$ $V_{IH}=V_{IHC}=V_{CC}-0.2\text{ V}$ $V_{IL}=V_{ILC}=0.2\text{ V}$		10	$\mu\text{A}$

(a) 電気的特性

番号	記 号	項 目	6 MHz 版		8 MHz 版		10 MHz 版	
			最小	最大	最小	最大	最小	最大
18	$T_{dIO(W/RW)}$	IORQ, CE の立ち下がりから W/RDY 立ち下がりまでの遅延(ウェイト・モード)		175		130		110
19	$T_{dC(W/RR)}$	クロックの立ち上がりから W/RDY 立ち下がりまでの遅延(レディ・モード)	100		95		85	
20	$T_{dC(W/RWz)}$	クロックの立ち下がりから W/RDY フロート状態までの遅延(ウェイト・モード)		110		90		80
22	$T_{wPA}$	高レベル・パルス幅	200		150		150	
23	$T_{wPI}$	低レベル・パルス幅	200		150		150	
24	$T_{CTXC}$	送信クロック周期	330		250		200	
25	$T_{wTxCi}$	低レベル送信クロック・パルス幅	100		85		80	
26	$T_{wTxCCh}$	高レベル送信クロック・パルス幅	100		85		80	
27	$T_{dTXC(TxD)}$	TxC の立ち下がりから TxD 信号までの遅延(SIO クロック×1 モード)		220		160		120
28	$T_{dTXC(W/RR)}$	TxC の立ち下がりから W/RDY 信号までの遅延(レディ・モード)	5	9	5	9	5	9
29	$T_{dTXC(INT)}$	TxC の立ち下がりから INT 立ち下がりまでの遅延	5	9	5	9	5	9
30	$T_{CRXC}$	受信クロック周期	330		250		200	
31	$T_{wRXCi}$	低レベル受信クロック・パルス幅	100		85		80	
32	$T_{wRXCCh}$	高レベル受信クロック・パルス幅	100		85		80	
33	$T_{SRxD(RxC)}$	RxC 立ち上がりに対する RxD 信号セットアップ時間(SIO クロック×1 モード)	0		0		0	
34	$T_{hRxD(RxC)}$	RxC 立ち上がりに対する RxD 信号ホールド時間(SIO クロック×1 モード)	100		80		60	
35	$T_{dRxC(W/RR)}$	RxC 立ち上がりから W/RDY 立ち下がりまでの遅延(レディ・モード)	10	13	10	13	10	13
36	$T_{dRxC(INT)}$	RxC の立ち上がりから INT 立ち下がりまでの遅延	10	13	10	13	10	13
37	$T_{dRxC(SYNC)}$	RxC 立ち上がりから SYNC 立ち下がりまでの遅延(出力モード)	4	7	4	7	4	7
38	$T_{SSYNC(RxC)}$	RxC の立ち上がりに対する SYNC 信号セットアップ時間(外部同期モード)	-100		-100		-100	

(b) Z80SIO の入出力端子のタイミング(単位: ns)

〈表 2〉  
SIO の各ポートの選択

B/A	C/D	セレクトされるポート	CPU の動作	SIO 動作
“L”	“L”	チャンネル A データ・ポート	リード	受信データ
			ライト	送信データ
“L”	“H”	チャンネル A コントロール・ポート	リード	リード・レジスタ (RR <sub>0</sub> ~RR <sub>1</sub> )
			ライト	ライト・レジスタ (WR <sub>0</sub> ~WR <sub>7</sub> )
“H”	“L”	チャンネル B データ・ポート	リード	受信データ
			ライト	送信データ
“H”	“H”	チャンネル B コントロール・ポート	リード	リード・レジスタ (RR <sub>0</sub> ~RR <sub>2</sub> )
			ライト	ライト・レジスタ (WR <sub>0</sub> ~WR <sub>7</sub> )

〈表 3〉 Z80SIO のライト/リード・レジスタ構成

レジスタ名	記号	役 目
ライト・レジスタ 0	WR <sub>0</sub>	コマンドおよびポインタ・ビット
ライト・レジスタ 1	WR <sub>1</sub>	割り込み制御および WAIT/READY の制御
ライト・レジスタ 2	WR <sub>2</sub>	割り込みベクトル(チャンネル B のみ)
ライト・レジスタ 3	WR <sub>3</sub>	レシーバ制御
ライト・レジスタ 4	WR <sub>4</sub>	コミュニケーション・モードの設定
ライト・レジスタ 5	WR <sub>5</sub>	トランスミット制御
ライト・レジスタ 6	WR <sub>6</sub>	シンク・キャラクタまたは従局アドレス(SDLC)
ライト・レジスタ 7	WR <sub>7</sub>	シンク・キャラクタまたはフラグ・パターン(HDLC)

(a) ライト・レジスタ

レジスタ名	記号	役 目
リード・レジスタ 0	RR <sub>0</sub>	割り込み要因の識別
リード・レジスタ 1	RR <sub>1</sub>	エラー状態の識別およびレシデュ・コード
リード・レジスタ 2	RR <sub>2</sub>	割り込みベクトル(チャンネル B のみ)

(b) リード・レジスタ

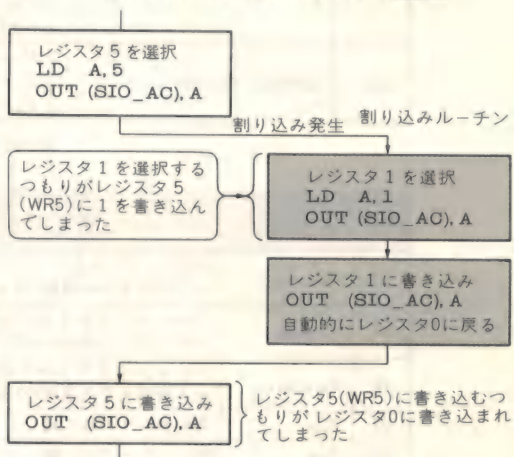
これらのリード/ライト・レジスタは、一部例外を除きチャンネル A およびチャンネル B それぞれ独立して存在しています。これらのレジスタに対するアクセスは、各チャンネルのコマンド・ポートに書き込み動作をすることでライト・レジスタへ書き込まれ、読み込み動作をすることでリード・レジスタより読み込まれます。

WR<sub>0</sub>~WR<sub>7</sub>、あるいは RR<sub>0</sub>~RR<sub>2</sub> まであるレジスタの切り替えは、レジスタ・ポインタを設定することでコントロール・ポートを一時的に切り替えるように使います。

リセット直後や読み書きシーケンスの終了状態では、コマンド・ポートに書き込みすると WR<sub>0</sub> に書き込まれ、読み込みをすると RR<sub>0</sub> の内容が読み込まれます。

ここで WR<sub>0</sub> に 01h を書き込むと、レジスタ・ポイン

〈図 4〉 SIO のレジスタ・アクセスの注意点



タは 1 になり、コントロール・ポートは WR<sub>1</sub> または RR<sub>1</sub> に切り替えられます。この次にコントロール・ポートに対して書き込み動作をすると WR<sub>1</sub> に書き込まれます。一度コントロール・ポートにアクセスすると、またレジスタ・ポインタはクリアされ、コントロール・ポートは WR<sub>0</sub> と RR<sub>0</sub> に戻ります。

### ● レジスタ・ポインタの注意

いちどアクセスするだけで、レジスタ・ポインタはクリアされてしまうので、WR<sub>2</sub> にデータを書き込んで RR<sub>2</sub> のデータを読むという場合でも、再度レジスタ・ポインタを 2 に設定しなければなりません。

またレジスタ・ポインタ設定中に割り込みが発生し、その割り込みルーチンでも SIO のレジスタを操作していた場合、図 4 のように割り込み処理終了後の SIO のレジスタに対するアクセスでは、意図しないレジスタに対して書き込みをしてしまう場合があります。

よってレジスタ・ポインタを設定して目的のレジスタの読み書きが終了するまでは、割り込み受け付け禁止状態にしておきます。

また読み込み動作時はリード・レジスタを読み込む



ので、ライト・レジスタに書き込んだ値が後で必要になっても読み出すことはできません。書き込み専用レジスタです。必要ならばワーク・エリア上に保存しておきます。

#### ● プログラミング上の注意点

SIO のプログラミングでとくに注意することは、書き込みレジスタ 4 (WR<sub>4</sub>) を他のレジスタより先に設定しなければならないことです。WR<sub>4</sub> は通信モードや送受信クロックの倍率を決定するレジスタで、これらの設定により送受信データやプログラミングが影響を受けるためです。特に同期通信では重要です。

また、SIO のレジスタにはチャンネル A にしかないものやチャンネル B にしかないものもあります。以下に各レジスタの詳細を示します。

### 書き込みレジスタの詳細

多機能さに比例して、レジスタの数も多い Z80SIO です。ここではレジスタの一覧をまとめて図 5 (pp. 116~117) に示します。

#### ● 書き込みレジスタ 0 (WR<sub>0</sub>)

レジスタ・ポインタと、基本動作コマンドの設定レジスタです。

##### ▶ D<sub>0</sub>~D<sub>2</sub>: レジスタ・ポインタ・ビット

これらのビットは、次のデータの書き込みまたは読み出しをするレジスタの番号を指定します。書き込みレジスタ 0 (WR<sub>0</sub>) に OUT 命令で出力した後に書き込み (OUT 命令を実行) すると、そのデータはここで指定された番号の書き込みレジスタ (WR<sub>0</sub>~WR<sub>7</sub>) に書き込まれます。また、出力した後に読み出し (IN 命令を実行) すると、ここで指定された番号の読み出しレジスタから読み込みます。

##### ▶ D<sub>3</sub>~D<sub>5</sub>: 基本コマンド・ビット

これらのビットは SIO の基本的な動作を指定するためのものです。

##### ・ 000b: ノー・オペレーション

何もしないコマンドです。このコマンドは単にレジスタ・ポインタをセットしたい場合に使います。例えばコマンド・ポートに 00000001b と書き込むと、通信動作に影響を与えることなく、レジスタ・ポインタを 1 (WR<sub>1</sub>/RR<sub>1</sub>) に設定します。

##### ・ 001b: アバート発生

HDLC モードにおいて、アバート・シーケンス (7 個の連続した “1”) を送信するためのコマンドです。データをどのように設定しても HDLC のゼロ・インサージョン機能により “1” が 5 個連続すると自動的に “0” を挿入するので、このコマンドはアバートを発生する唯一の方法です。

##### ・ 010b: 外部/ステータス割り込みのリセット

外部/ステータス (CTS 端子, DCD 端子, シンク/ハント, 送信アンダーラン/EOM) が変化した場合に、SIO は RR<sub>0</sub> をラッチします。よって RR<sub>0</sub> で読み込むことのできる外部/ステータスは、その時点での最新の状態ではなく、最初にどれかが変化したときラッチされた状態となります。よって今の状態を知りたいとき、または外部/ステータス割り込み処理ルーチンの中で、次の変化で再度割り込みを発生させる場合はこのコマンドを実行しなければなりません。

##### ・ 011b: チャンネル・リセット

書き込まれたチャンネルのすべての状態を RESET 端子によりリセットされたのと同様に初期化します。RESET 端子によるリセットと異なり、書き込まれたチャンネルしかリセットしませんから、ソフトウェアにより SIO を完全にリセットしたい場合はチャンネル A とチャンネル B の両方に対してチャンネル・リセット (18h) を書き込まなくてはなりません。

なお、チャンネル A に対するチャンネル・リセットは割り込み優先回路をもリセットします。

##### ・ 100b: 次のキャラクタ受信時の割り込みをイネーブル

このコマンドは、書き込みレジスタ 1 (WR<sub>1</sub>) のビット 3~4 で最初のキャラクタ受信時に割り込みが選択されている状態で、最初のキャラクタ受信の割り込みを処理した後に、次のキャラクタ受信で割り込みを発生させたい場合に実行します。詳しくは後述します。

##### ・ 101b: 保留中の送信割り込みをリセット

送信割り込みがイネーブルのとき、送信バッファがエンプティ (空) になると割り込みが発生します。このコマンドは、その際に送るべきデータがもうない場合に実行すると、それ以後は送信割り込みは発生しなくなります。

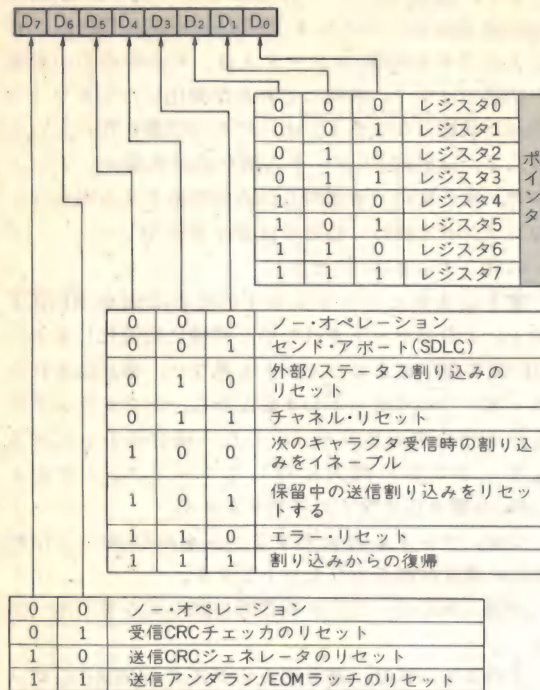
##### ・ 110b: エラー・リセット

パリティ・エラーまたはオーバーラン・エラーが発生すると、読み出しレジスタ 1 (RR<sub>1</sub>) のビット 4 と 5 にラッチされたままになります。この機能はデータをブロック受信する際に、最後にエラー・ステータスを読めばそのブロックにエラーがあったかどうか判別できるように気を利かせたものですが、逆に 1 キャラクタごとにエラー・チェックをする場合には、そのたびにエラー・リセットをしなくてはなりません。注意が必要です。

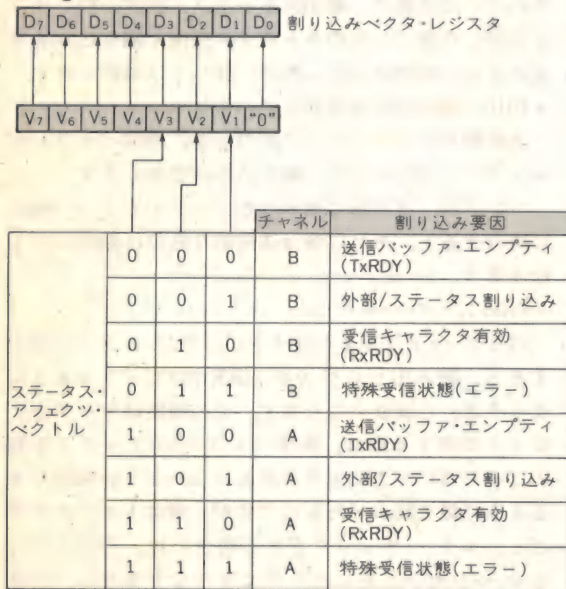
##### ・ 111b: 割り込みからの復帰

このコマンドは Z80 以外の CPU で Z80SIO を使うときに、RETI 命令の実行により割り込みデジャ・チェーンの復帰をする手段として用意されています。しかし実際には Z80SIO を Z80 以外の CPU に接続するには外部回路が大きくなりすぎます。ほぼ同機能の LSI である  $\mu$ PD7201A や 72001A (どちらも NEC) な

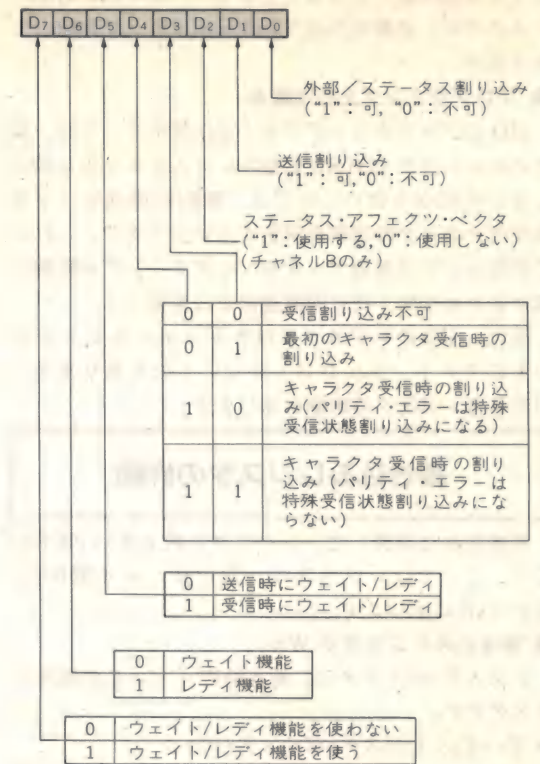
► WR<sub>0</sub>



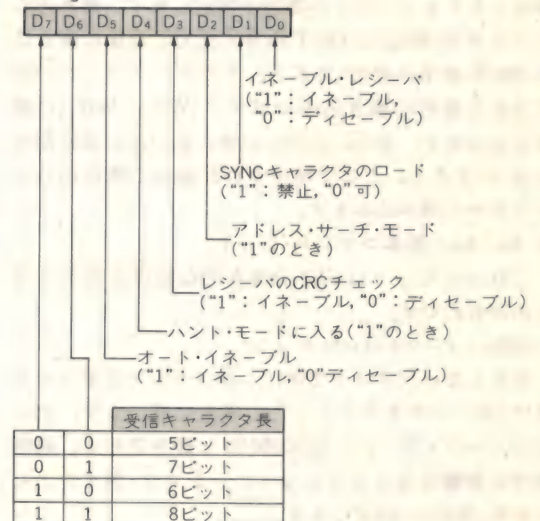
► WR<sub>2</sub>



► WR<sub>1</sub>



► WR<sub>3</sub>



どを使うのが現実的ですので、あまり使われません。

► D<sub>6</sub>~D<sub>7</sub>: CRC リセット・コマンド

この2ビットで受信CRCチェッカのリセット, 送信CRCジェネレータのリセット, EOMのセット(HDLCモードでFCSおよび終了フラグを送信する)が行えます。

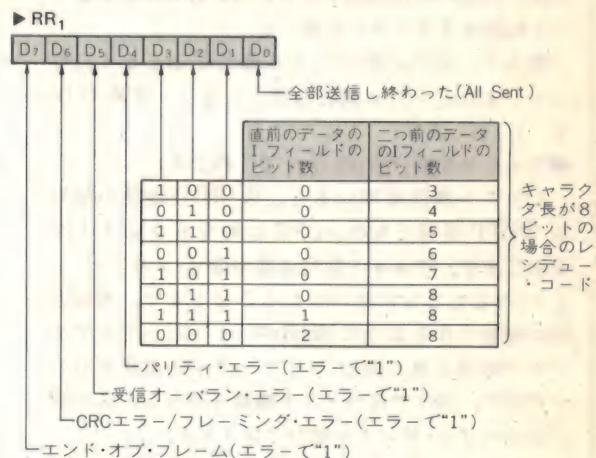
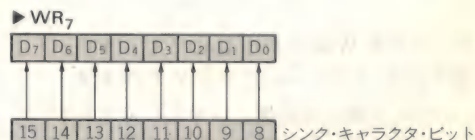
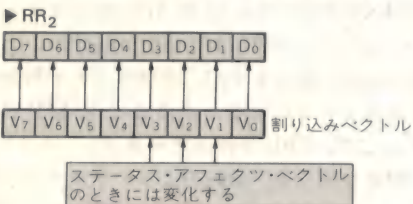
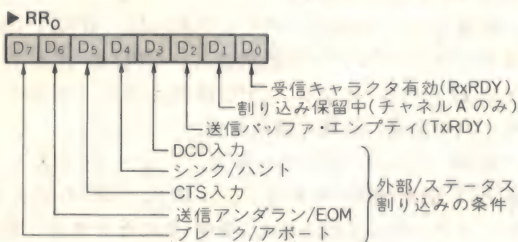
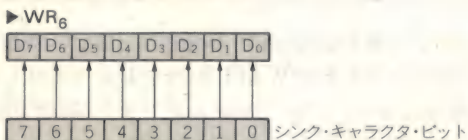
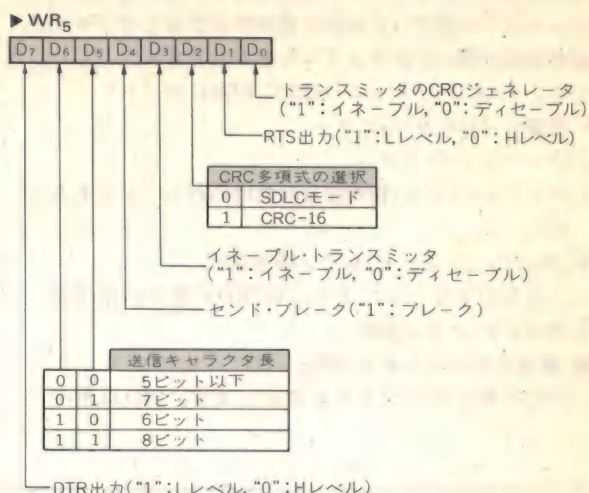
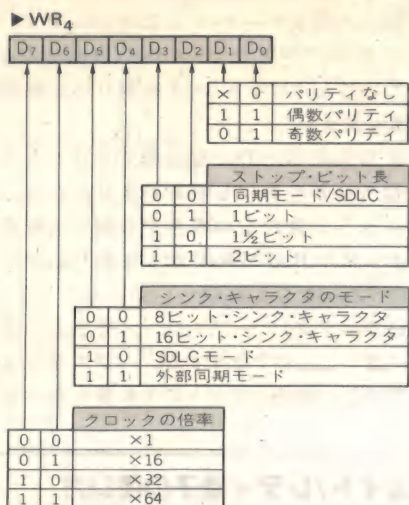
● 書き込みレジスタ1(WR<sub>1</sub>)

WR<sub>1</sub>は割り込みとウェイト/レディの設定をします。

► D<sub>0</sub>: 外部/ステータス割り込みイネーブル

このビットがセットされていると、非同期モードでブレイク信号(連続したスペース状態=Lレベル)やHDLCモードでアポート・シーケンスを受信した場合、





CTS 端子や DCD 端子, SYNC 端子の状態が変化した場合, 送信アンダーラン/HDLC モードでのエンド・オブ・フレーム(EOM)で外部/ステータス割り込みを発生します。

#### ► D<sub>1</sub>: 送信割り込みイネーブル

このビットがセットされていると, 送信バッファがエンプティ(空)になると送信割り込みを発生します。

#### ► D<sub>2</sub>: ステータス・アフェクツ・ベクタ

このビットがセットされていると, 割り込み発生時に割り込みチャンネルや要因の種類により WR<sub>2</sub>に設定されている割り込みベクタ(下位8ビット)の D<sub>1</sub>~D<sub>3</sub>を変化させて, 割り込みベクタとして出力します。この機能は割り込み要因によりベクタを変えることで,

割り込み処理ルーチンの中で要因を判別する手間を省き, 処理を高速に行うためのもので, CPUが割り込みモード2で使われている場合に有効です。

#### ► D<sub>3</sub>~D<sub>4</sub>: 受信割り込みモード

これらのビットは, 受信割り込みのモードを決めます。

- 00b: 受信割り込みを使用しない
  - 01b: 最初のキャラクタ受信時のみ割り込みを発生
  - 10b: キャラクタ受信割り込み(パリティ・エラーは特殊受信状態割り込み)
  - 11b: キャラクタ受信割り込み(パリティ・エラーは特殊受信状態割り込みにならない)
- パリティ・エラーを除いて, 基本的にはエラーが発

生したときは、キャラクタ受信割り込みは発生しません。エラーの発生した状態を特殊受信状態と呼びます。特殊受信状態には次のようなものがあります。

- ・エンド・オブ・フレーム (HDLC/SDLC モード)
- ・受信オーバーラン・エラー
- ・フレーミング・エラー
- ・パリティ・エラー (特殊受信状態にいけないときもある)

#### ▶ D<sub>5</sub>~D<sub>7</sub>: ウェイト/レディ機能選択

これらの3ビットにより、 $\overline{W/RDY}$  端子の使用法を決めます(コラム参照)。

#### ● 書き込みレジスタ 2(WR<sub>2</sub>)

WR<sub>2</sub>は割り込みベクタを設定します。SIO は割り

込みを発生させ、CPU より割り込みベクタの要求があると、WR<sub>2</sub>の内容をデータ・バスに出力します。このときステータス・アフェクト・ベクタ(WR<sub>1</sub>のD<sub>2</sub>)がセットされていると、ビット1~3を割り込み要因に応じて変化させます。

この場合はWR<sub>2</sub>のD<sub>1</sub>~D<sub>3</sub>の値は無効になります。また、WR<sub>2</sub>はチャンネルBにしかありませんから、たとえチャンネルAしか使わない場合でも割り込みを使う場合は、チャンネルBのWR<sub>2</sub>に対して割り込みベクタを設定しなければなりません。

また割り込みベクタ・テーブルは、メモリ上の任意のアドレスに置くことができますが、ベクタを生成する際に加算ではなく単純にビットの入れ替えをするだ

## ウェイト/レディ端子の使い方

書き込みレジスタ WR<sub>1</sub>のD<sub>5</sub>~D<sub>7</sub>の3ビットと、 $\overline{W/RDY}$  端子の使い方についてまとめてみます。これらのレジスタや端子は通常では使用しませんが、外部に独自の転送機構を付けたり、DMA によるデータ転送をするときに必要です。

表Aに、WR<sub>1</sub>の設定による動作を示します。ウェイトまたはレディ機能を使うときは、WR<sub>1</sub>のD<sub>7</sub>を“1”にします。

#### ●ウェイト機能(WR<sub>1</sub>のD<sub>5</sub>=“0”のとき)

ウェイト機能を用いると、 $\overline{W/RDY}$  端子の信号は $\overline{WAIT}$ 信号となり、CPU に対するウェイトを要求します。ウェイト信号は他の信号(メモリ・ウェイト)などとともにより用いられることもあり、周辺回路が簡単になるように(要因がいくつあってもプルアップ抵抗1本で済む)ワイヤードOR接続するのが普通で、SIO ではウェイト機能を使うと、この端子はオープン・ドレイン出力になります。

ウェイト機能が指定されているとき、WR<sub>1</sub>のD<sub>5</sub>が“0”であるとトランスミッタに対して $\overline{WAIT}$ 信号が用いられます。つまり、SIO のデータ・レジ

スタに対して書き込み(送信)があったときに、送信バッファが空になるまで $\overline{WAIT}$ 出力をLレベルにしてCPUを停止させ、バッファがエンプティ(送信可能)になるのを待たせます。

送信バッファがエンプティであれば、 $\overline{WAIT}$ 出力はオープン(プルアップ抵抗によりCPUの $\overline{WAIT}$ 端子はHレベル)になり、CPUは何もなかったかのように次の命令の処理に進みます。

WR<sub>1</sub>のD<sub>5</sub>が“1”であると、レシーバに対して $\overline{WAIT}$ 信号が用いられます。つまり、SIO のデータ・レジスタに対して読み出し(受信)されたときに、受信データ有効(RxRDY)であれば $\overline{WAIT}$ 出力はオープン(Hレベル)になり、CPUは何もなかったかのように次の命令の処理に進みますが、受信データが有効でなかったときはデータが受信されるまで $\overline{WAIT}$ 出力をLレベルにして、CPUを停止させます。

ウェイト機能を使うと、前記のようにプログラムで送信バッファ・エンプティ(TxRDY)や受信データ有効(RxRDY)ビットをポーリング(“1”になるまで待つ)することなく、データの送信または受信ができま

〈表A〉ウェイト/レディ端子の使い方

WR <sub>1</sub>			$\overline{W/RDY}_A$ , $\overline{W/RDY}_B$ の機能	H/L レベル	状 態
D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>			
1	0	0	$\overline{WAIT}$ (TxRDY)	L	トランスミッタがバッファ・エンプティでない(送信できない)
				フローティング	トランスミッタがバッファ・エンプティ (送信できる)
1	0	1	$\overline{WAIT}$ (RxRDY)	L	レシーバにデータが準備できていない (受信できない)
				フローティング	レシーバにデータが準備できている (受信できる)
1	1	0	$\overline{RDY}$ (TxRDY)	L	トランスミッタがバッファ・エンプティ (送信できる)
				H	トランスミッタがバッファ・エンプティでない(送信できない)
1	1	1	$\overline{RDY}$ (RxRDY)	L	レシーバにデータが準備できている (受信できる)
				H	レシーバにデータが準備できていない (受信できない)



けなので、16 バイト(8 個のベクタ×2 バイト・ベクタ)が繰り上がるようなアドレス(xxFlh~xxFFh)を開始アドレスにすることはできません。

このため当然のことながら、チャンネル A と B でまったく別々のアドレスに割り込みベクタ・テーブルを設定することはできません。

### ● 書き込みレジスタ 3(WR<sub>3</sub>)

WR<sub>3</sub>はレシーバの機能を設定します。

#### ▶ D<sub>0</sub>: イネーブル・レシーバ

このビットが“1”であるとレシーバがイネーブルされ受信動作が可能になります。ただし、オート・イネーブル(WR<sub>2</sub>の D<sub>6</sub>)がセットされていると、 $\overline{\text{DCD}}$  入力端子がアクティブ(L レベル)でないとデータは受信

されません。このビットは受信に関するすべての設定が完了した後にセットします。

#### ▶ D<sub>1</sub>: SYNC キャラクタのロード禁止

このビットが“1”にセットされているとき、同期モードにおいて SYNC キャラクタ(WR<sub>6</sub>, WR<sub>7</sub>に設定されているデータ)、HDLC モードにおいてフラグ(WR<sub>7</sub>に設定されているデータ)を受信しても受信 FIFO にはロードされません。

つまり、これらのデータをデータ・レジスタに受信されないようにします。

この機能は同期または HDLC モードで、受信データから SYNC キャラクタまたはフラグ・シーケンスを除外する手間を省くためのものです。

ですが、 $\overline{\text{WAIT}}$  出力が L レベルの間は CPU を停止させられてしまうので、割り込みを受け付けなくなります。

ここでは CPU をデータの送信または受信のどちらかだけで使う場合を除いては、あまり使いものになりません。送信または受信のみで使うにしてもプログラムが IN 命令または OUT 命令を繰り返していないと送受信できませんから、CPU は送受信以外の処理を行うことができなくなります。もしも、高速通信が必要な場合は次に述べるレディ機能を使い DMA 転送で行うほうが得策です。

### ● レディ機能(WR<sub>1</sub>の D<sub>6</sub>=“1” のとき)

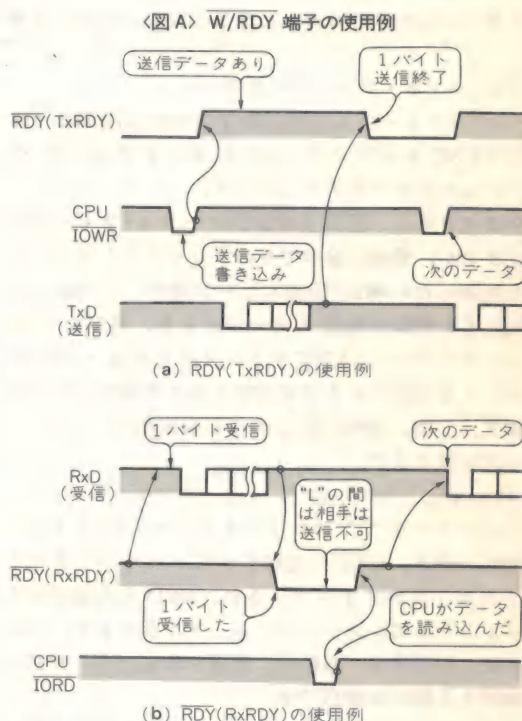
レディ機能を用いると、 $\overline{\text{W/RDY}}$  端子は DMA に対するレディ出力になります。

WR<sub>1</sub>の D<sub>6</sub>が“0”のときはレディ出力はトランスミッタに対して用いられ、送信バッファ・エンプティ(TxRDY)になると L レベルになり、送信バッファ・エンプティでないと H レベルになります [図 A (a)]。

WR<sub>1</sub>の D<sub>6</sub>が“1”のときはレディ出力はレシーバに対して用いられ、受信データ有効(RxRDY)になると L レベルになり、受信データ有効でないと H レベルになります。

これらのレディ出力を Z80DMA の RDY 入力端子に接続して、L レベルでアクティブになるように DMA をプログラムしておけば、TxRDY になったときに自動的に DMA によりメモリのパッファからデータが送信され、または、RxRDY になったときに自動的に DMA により受信データがメモリのパッファに受信されるようになります。

送信と受信は同時に DMA による転送はできませんが、例えば送信を割り込みで、受信を DMA で転送す



ることにより、CPU は送受信のための SIO のポーリングから開放されます。

またレディ機能は DMA 以外にも、シリアル入出力対応のワンチップ・マイコンなどとの通信のとき、1 バイトごとのハンドシェイク信号としても使用できます。

#### ▶ D<sub>2</sub>: アドレス・サーチ・モード

このビットは、HDLC モードで従局(二次局)をプログラムする際に、WR<sub>6</sub>に設定されたアドレスまたはグローバル・アドレス (FFh) をもつフレーム以外を無視したいときに“1”にセットします。

HDLC の従局では自分のアドレス以外のフレームを無視しなければならない、この機能は関係ないフレームでの受信割り込みまたは受信データの発生を阻止し、CPU が効率よく他の処理ができるように設けられたものです。

#### ▶ D<sub>3</sub>: レシーバ CRC イネーブル

このビットがセットされていると、受信シフトレジスタから受信 FIFO に最後の受信データ・ビットが転送されるときに CRC の計算を始めます。BSC などの通信プロトコルではブロックの途中から CRC 計算を始めなければならない、開始位置も透過モードと非透過モードで異なりますから、このビットはデータの受信中にセットしたりリセットしたりしなくてはなりません。

#### ▶ D<sub>4</sub>: ハント・フェーズに入る

ハント・フェーズとは同期確立のために同期モードでは SYNC キャラクタ、HDLC モードではフラグ・シーケンスをサーチすることです。

SIO はリセット直後に自動的にハント・フェーズに入りますが、受信の途中でそのブロックまたはフレームを廃棄したい場合(伝送エラーが発生してそれ以上受信しても無駄な場合)にセットすると、再びハント・フェーズに入り、SYNC キャラクタまたはフラグ・シーケンスを受信するまで受信割り込みを発生しなくなります。なお、同期が確立すると自動的にハント・フェーズを抜けます。

#### ▶ D<sub>5</sub>: オート・イネーブル

このビットを“1”にセットするとオート・イネーブル機能が働き、CTS 入力端子が L レベルのときにトランスミッタがイネーブルされ、 $\overline{\text{DCD}}$  入力端子が L レベルのときにレシーバがイネーブルされます。これは必要な制御信号が自動的に監視されるため、モデムを接続する際には便利です。

このビットがセットされていると、CS がオフのときは送信が自動的に止まり、CD がオフのとき(つまりキャリアが届いていないとき)の受信データはノイズによるものなので自動的に無視するようになります。ハードウェア・フロー制御(RS-CS フロー)のときに便利です。

#### ▶ D<sub>6</sub>~D<sub>7</sub>: 受信キャラクタ長

これら 2 ビットにより受信キャラクタのビット長を設定します。なお、受信キャラクタ長が 7 ビット以下の場合にはデータは右詰め(下位ビット側)でデータ・レジスタにロードされ、その左にパリティ・ビットが付

加されるので、もとのデータを再現するためにはパリティ・ビットをマスクしなければなりません。

#### ● 書き込みレジスタ 4 (WR<sub>4</sub>)

このレジスタはトランスミッタとレシーバの両方に共通なパラメータを設定します。

#### ▶ D<sub>0</sub>: パリティ・イネーブル

このビットを“1”にセットすると、トランスミッタのパリティ・ジェネレータおよびレシーバのパリティ・チェッカがイネーブルされ、送信データにパリティ・ビットが付加され、受信データのパリティ・ビットがチェックされます。

#### ▶ D<sub>1</sub>: 偶数/奇数パリティの選択

このビットを“1”に設定すると偶数、“0”に設定すると奇数パリティとなります。なお、パリティ・イネーブルがディセーブルのとき(WR<sub>4</sub>の D<sub>0</sub>が“0”)はこのビットは“1”でも“0”でも無効です。

#### ▶ D<sub>2</sub>~D<sub>3</sub>: ストップ・ビット長

これらのビットにより非同期モードでのストップ・ビット長を設定します。また、同期モードおよび HDLC モードの場合は、両方とも“0”に設定する必要があります。ストップ・ビット長は大は小を兼ねる特徴があり、送信データのストップ・ビットが長くても単にキャラクタの間隔とみなされ(伝送効率は悪くなる)、受信データのストップ・ビットが足りなくてもパリティを使わない場合はエラーは発生しません。見落としやすい点ですから注意しましょう。

#### ▶ D<sub>4</sub>~D<sub>5</sub>: 同期モード

これらのビットは同期モード(WR<sub>4</sub>の D<sub>2</sub>=“0”, D<sub>3</sub>=“0”)のときの同期モードの種類を設定します。

#### ▶ D<sub>6</sub>~D<sub>7</sub>: 送受信クロックの倍率

これらのビットは送受信クロック(TxC, RxC 入力端子)のデータ送受信レートに対する倍率を設定します。同期モードおよび HDLC モードでは 1 倍に設定しなければなりません。また非同期モードでは原則的に 16 倍以上を設定しなければなりません。

#### ● 書き込みレジスタ 5 (WR<sub>5</sub>)

WR<sub>5</sub>はトランスミッタのパラメータを設定します。

#### ▶ D<sub>0</sub>: トランスミッタ CRC イネーブル

このビットが“1”にセットされていると、トランスミッタの CRC ジェネレータがイネーブルされます。送信データが送信 FIFO から送信シフトレジスタにロードされたときにこのビットが“1”であるとそのデータに対して CRC が計算されます。また、このビットが“1”にセットされていて、同期モードまたは HDLC モードのときに送信アンダーラン(送信すべきデータがない)が発生すると、自動的に CRC が送信されます。

#### ▶ D<sub>1</sub>: RTS 端子制御

このビットが“1”にセットされていると  $\overline{\text{RTS}}$  端



子に L レベルが, “0” にセットされると H レベルが出力されます。ただし非同期モードのとき, “0” にセットしていても, 送信バッファに送信すべきデータがあると L レベルになり, 送信バッファが空になってから H レベルになります。

これは RTS 端子の本来の動作で, 送信すべきデータがあるとき L レベルになります。現在の全二重通信では, この端子を RS-CS フロー制御などに使い, 受信バッファがオーバーしそうなとき H レベルにして相手の送信を止める意味でも使われています。

#### ▶ D<sub>2</sub>: CRC 多項式の選択

このビットが “0” に設定されていると CRC は CCITT ( $X^{16}+X^{12}+X^5+1$ ) が, “1” に設定されていると CRC16 ( $X^{16}+X^{15}+X^2+1$ ) が選択されます。BSC では CRC16 が, HDLC では CCITT 方式が一般的です。

#### ▶ D<sub>3</sub>: トランスミッタ・イネーブル

このビットが “1” にセットされているとトランスミッタがイネーブルされます。データの送信中に “0” に設定しても送信が終わるまではイネーブルのままです。

オート・イネーブルがセットされている状態で実際に送信するためには, CTS 入力端子が L レベルでな

ければなりません。

#### ▶ D<sub>4</sub>: ブレーク信号の送信

このビットを “1” にセットすると, どのようなデータを送信していても送信データ (TxD) は強制的にスペース (“0”=L レベル) にされます。

一般的にブレーク信号は相手の送信を強制的に停止させたい場合に用いられます。実際にブレーク信号として機能させるためには, キャラクタ長+スタート・ビット+ストップ・ビット長よりも長くスペース状態を維持する必要があります。

#### ▶ D<sub>5</sub>~D<sub>6</sub>: 送信キャラクタ長

これらのビットは送信データのビット長を設定します。5 ビット以下の設定の場合は, 実際のデータが何ビットであるかを SIO が判別できるように表 4 のように実際のビットの左側に “000” を付加し, さらにその左側に (5-実際のビット数) の “1” を付加します。

#### ▶ D<sub>7</sub>: DTR 端子制御

DTR 端子の状態を設定します。“1” を書き込むと L レベルが, “0” を書き込むと H レベルが出力されます。この端子は非同期モードでも, 送信バッファの状態に関係なくいつでも自由に設定できます。

#### ● 書き込みレジスタ 6 (WR<sub>6</sub>)

WR<sub>6</sub> には同期モードでは SYNC キャラクタ (バイ

## SIO の送受信クロック倍率

SIO の送受信クロック倍率は, 16 倍/32 倍/64 倍の 3 通りの設定が可能ですが, 実際にはどれを使ったら良いのか迷ってしまいます。とくに CTC と組み合わせる場合には CTC の設定によってもソフトウェア的に変更できるのでなおさらです。

SIO の送受信クロック倍率は, 理論的には大きいほうが受信マージン (受信データ・ビットのひずみに対する余裕度) が大きくなって良いのですが, 実際にはどちらでもほとんど違いはありません。

そこで, 最高データ・レートと最小データ・レートを考慮して決定するのが基本になります。

現在使われるであろう最低は 300 bps で, 最高は速ければ速いほど良いと思われます。RS-232-C 規格の最高は 20 kbps ですが, 最近はこれよりも高速なモデムもありますから, やはり最高は速ければ速いほど良いでしょう。

参考までに, CPU クロックと使うことができる通信速度を表にまとめておきます。表中でアミがかかっている部分は, 周波数が正確に合っているわけではないので, 同期モードでは使用できません。

〈表 B〉 通信速度と CTC の設定

通信速度	CPU クロック					
	4 MHz	4.9152 MHz	6 MHz	8 MHz	9.8304 MHz	10 MHz
38400 bps	×	×	×	×	1	×
28800 bps	×	×	×	×	×	×
19200 bps	×	1	×	×	2	×
14400 bps	×	×	×	×	×	×
9600 bps	×	2	×	×	4	×
4800 bps	×	4	×	×	8	×
2400 bps	×	8	×	13	16	16
1200 bps	13	16	×	26	32	33
600 bps	26	32	39	52	64	65
300 bps	52	64	78	104	128	130
200 bps*	78	96	117	156	192	195
150 bps*	104	128	156	208	256 (0)	△
110 bps*	142	174	213	△	△	△
75 bps*	208	256 (0)	△	△	△	△
50 bps*	×	△	△	△	△	△

\*ほとんど使われない

数字: CTC タイマ・モード, プリスケアラ 16 で SIO クロック倍率 16 のときの CTC 分周比

( ) 内はレジスタに書き込む値

△: SIO クロック 32 倍クロック設定で可能

× CTC のタイマ・モード (プリスケアラ 16 または 256 分周) では不可

シンクの場合は1バイト目)を設定します。設定されたデータはSYNCキャラクタのロード禁止(WR<sub>3</sub>のD<sub>1</sub>が“1”)の場合は受信データにはなりません。しかし、送信時にはプログラムで送信データ・レジスタに書き込まなければなりません。

HDLCモードでアドレス・サーチ・モード(WR<sub>3</sub>のD<sub>2</sub>が“1”)の場合、従局アドレスを設定しておく、そのアドレスまたはグローバル・アドレス(FFh)以外のアドレスをもつフレーム以外は受信データになりません。非同期モードでは設定する必要はありません。

#### ● 書き込みレジスタ7(WR<sub>7</sub>)

WR<sub>7</sub>には同期モードでバイシンク(16ビット SYNCキャラクタ)の場合に2バイト目のSYNCキャラクタを設定します。しかし、送信時にはプログラムで送信データ・レジスタに書き込まなければなりません。HDLCモードではフラグ・シーケンスのパターン(7Eh)を設定します。

HDLCモードではトランスミッタ・イネーブルのときに送信するデータがなくなると、(送信CRCイネーブルのときはCRCに続いて)自動的にフラグ・シーケンスが送信されます。

### 読み出しレジスタの詳細

#### ● 読み出しレジスタ0(RR<sub>0</sub>)

RR<sub>0</sub>はSIOの基本的な状態を表します。

##### ▶ D<sub>0</sub>: 受信キャラクタ有効

このビットは受信FIFOに1キャラクタ以上の受信データがあるときに“1”になります。CPUが1キャラクタを読み込んでも、受信FIFOにまだデータがある場合には“0”になりません。

##### ▶ D<sub>1</sub>: 割り込み保留中

このビットはSIOが割り込みを発生する何らかの状態をもっているときに、割り込みの種類に関係なく“1”になります。なお、このビットはチャンネルAのRR<sub>0</sub>にしかありませんので、チャンネルBの割り込み保留中を確認する場合でもチャンネルAのRR<sub>0</sub>を読み込まなければなりません。

##### ▶ D<sub>2</sub>: 送信バッファ・エンプティ

このビットは送信バッファ(FIFO)に空きができると“1”になります。SIOはリセットされるとこのビットが“1”になります。しかし、送信データを書き込んでもSIOの初期化がされていないと(オート・イネーブルでCTS端子がHレベルのときも)永久に“0”になったままです。

##### ▶ D<sub>3</sub>: DCD(データ・キャリア検出)の状態

このビットは最初に変化した後のDCD入力端子の状態を表します。DCD端子がHレベルからLレベルに変化すると、このビットは“1”になり、Lレベル

〈表4〉送信データが5ビット以下の場合

キャラクタ当たりの 送信ビット数 (キャラクタ長)	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1	1	1	1	1	0	0	0	D <sub>0</sub>
2	1	1	1	0	0	0	D <sub>1</sub>	D <sub>0</sub>
3	1	1	0	0	0	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
4	1	0	0	0	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
5	0	0	0	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

D<sub>n</sub>: データ・ビット

からHレベルに変化すると“0”になります。以後、外部/ステータス割り込みのリセット・コマンド(WR<sub>0</sub>に10hを書き込む)が実行されるまで、この状態はラッチされ変化しません。したがって、最新の状態を読み込みたいときは、外部/ステータス割り込みのリセット・コマンドを実行する必要があります。このラッチ動作は以下のD<sub>4</sub>からD<sub>7</sub>にも適用されます。

##### ▶ D<sub>4</sub>: シンク/ハント

このビットはモードにより意味が異なります。

非同期モードではSYNC端子の状態を表します。

内部同期モードではエンター・ハント・フェーズ(WR<sub>3</sub>のD<sub>4</sub>をセット)によりセットされ、SIOが同期キャラクタ(モノシンクのときはWR<sub>0</sub>, バイシンクのときはWR<sub>0</sub>とWR<sub>7</sub>が連続して)が受信されたときにリセットされて“0”になります。

HDLCモードではエンター・ハント・フェーズによりセットされ、SIOがオープニング・フラグ(WR<sub>7</sub>のデータ=7Eh)が受信されたときにリセットされて“0”になります。

外部同期モードでは外部に設けられた同期検出回路の出力をSYNC端子に入力しますが、そのSYNC端子が変化したときに変化後の状態を表します。

##### ▶ D<sub>5</sub>: CTS(クリア・ツー・センド)の状態

このビットは最初に変化した後のCTS入力端子の状態を表します。CTS端子がHレベルからLレベルに変化すると、このビットは“1”になります。LレベルからHレベルに変化すると“0”になります。しかし、変化するのは最初の1回のみで、最新の状態を調べたい場合は外部/ステータス割り込みのリセット・コマンドを実行する必要があります。

##### ▶ D<sub>6</sub>: 送信アンダーラン/EOM

SIOがリセットされると、このビットは“1”にセットされます。このビットをリセットする唯一の方法はリセット・トランスミッタ・アンダーラン・コマンドを実行する(WR<sub>0</sub>にC0hを書き込む)ことです。また、送信FIFOが空になっても送信データを書き込まないと、送信アンダーランが発生してこのビットがセットされ、同時に外部/ステータス割り込みが発生します。



同期モードではこのビットがセットされていると、送信アンダーランが発生しても外部/ステータス割り込みを発生せず、アイドル・シンク・キャラクタ (WR<sub>6</sub> のデータ) を送信して同期確立を維持します。同期モードでこのビットがリセットされていると、送信 FIFO にデータがなくなると 2 バイトの CRC を送信し、外部/ステータス割り込みを発生します。BSC などでは CRC を送信したいときはデータ・ブロックの最後 (ETX 送信後など) でこのビットをリセットしておけば自動的に CRC が送信されます。

読み出しレジスタの CPU への読み込みは IN 命令を使います。

HDLC モードではこのビットがセットされていると、送信アンダーランが発生しても外部/ステータス割り込みを発生せずに、フラグ・シーケンス (WR<sub>7</sub> のデータ = 7Eh) を送信し続けて同期を維持します。しかし、HDLC ではフレームの途中でフラグ・シーケンスが挿入されると、それをクロージング・フラグと解釈して無効フレーム (ビット数が少ないか CRC に誤りがある) と判断され廃棄されますので、送信アンダーランを起こさないように送信しなければなりません。どうしても送信を中断しなくてはならない場合にはアボート・シーケンスを送信しなくてはなりません。

HDLC モードでこのビットがリセットされていると、送信アンダーランが発生した時点でフレームの終わりで見なされて CRC (FCS) が自動的に送信され、同時に外部/ステータス割り込みが発生します。CRC の送信が終了すると送信バッファ・エンプティ割り込みが発生しますが、そこで保留中の送信割り込みリセット・コマンド (28h) を WR<sub>0</sub> に書き込むと、それ以降は送信バッファ・エンプティ割り込みは発生せず、送信データを書き込まずにいればクロージング・フラグが送信され続けます。

HDLC 回線を不活性化 (フラグ・シーケンスを常に送らないで “1” に固定する) にはトランスミッタ・イネーブル (WR<sub>5</sub> の D<sub>5</sub>) を “0” にします。

#### ▶ D<sub>7</sub>: ブレーク/アボート検出

非同期モードでブレーク信号を受信するとセットされ、外部/ステータス割り込みを発生します。また、外部/ステータス割り込みをリセットしておくと、ブレーク信号が解除されたときにも外部/ステータス割り込みを発生します。

HDLC モードでアボート・シーケンスを受信するとセットされ、外部/ステータス割り込みを発生します。また、外部/ステータス割り込みをリセットしておくと、アボート状態が解除されたときにリセットされ、外部/ステータス割り込みを発生します。

#### ● 読み出しレジスタ 1 (RR<sub>1</sub>)

このレジスタには特殊受信状態割り込みの原因など

が設定されます。

#### ▶ D<sub>0</sub>: 全部送信し終わった

非同期モードにおいて送信 FIFO の全データを送信し終わると “1” になります。同期モードおよび HDLC モードでは常に “1” になります。半二重通信で送信から受信に切り替える (RS をオフにする) 前に、このビットが “1” になるのを確認します。

#### ▶ D<sub>1</sub>~D<sub>3</sub>: レシデュエ・コード

HDLC では伝送するデータはキャラクタ単位ではなくビット単位に可能で、たまたま 8 ビットごとに伝送しているだけです。I フレームを受信した場合の最後の 8 ビットはすべて有効なビットとは限りません。

そこで、最後のデータとその直前のデータの有効ビット数をこれらのビットで表します。これらのビットが意味をもつのはエンド・オブ・フレーム (RR<sub>1</sub> の D<sub>7</sub> が “1”) のときだけです。

#### ▶ D<sub>4</sub>: パリティ・エラー

このビットはパリティがイネーブルされているときに受信データ中にパリティ・エラーが検出されると “1” にセットされます。いったんセットされるとエラー・リセット・コマンドが実行 (WR<sub>0</sub> に 30h を書き込む) されるまでそのままです。したがって、データ・ブロックの受信が終わってからエラー・チェックをすることが可能です。

#### ▶ D<sub>5</sub>: オーバーラン・エラー

受信 FIFO がいっぱいになると、さらに次のデータが受信されると “1” になります。ただし実際にエラーがセットされるのは、実際にオーバーランが発生したデータが読み込まれたときにセットされます。このビットがセットされたときは受信データの CPU への取り込みが間に合わず受信データを失った、つまり受信データは相手が送ったデータより少ないことを意味します。

いったんセットされるとエラー・リセット・コマンドが実行されるまでそのままです。

#### ▶ D<sub>6</sub>: CRC/フレーミング・エラー

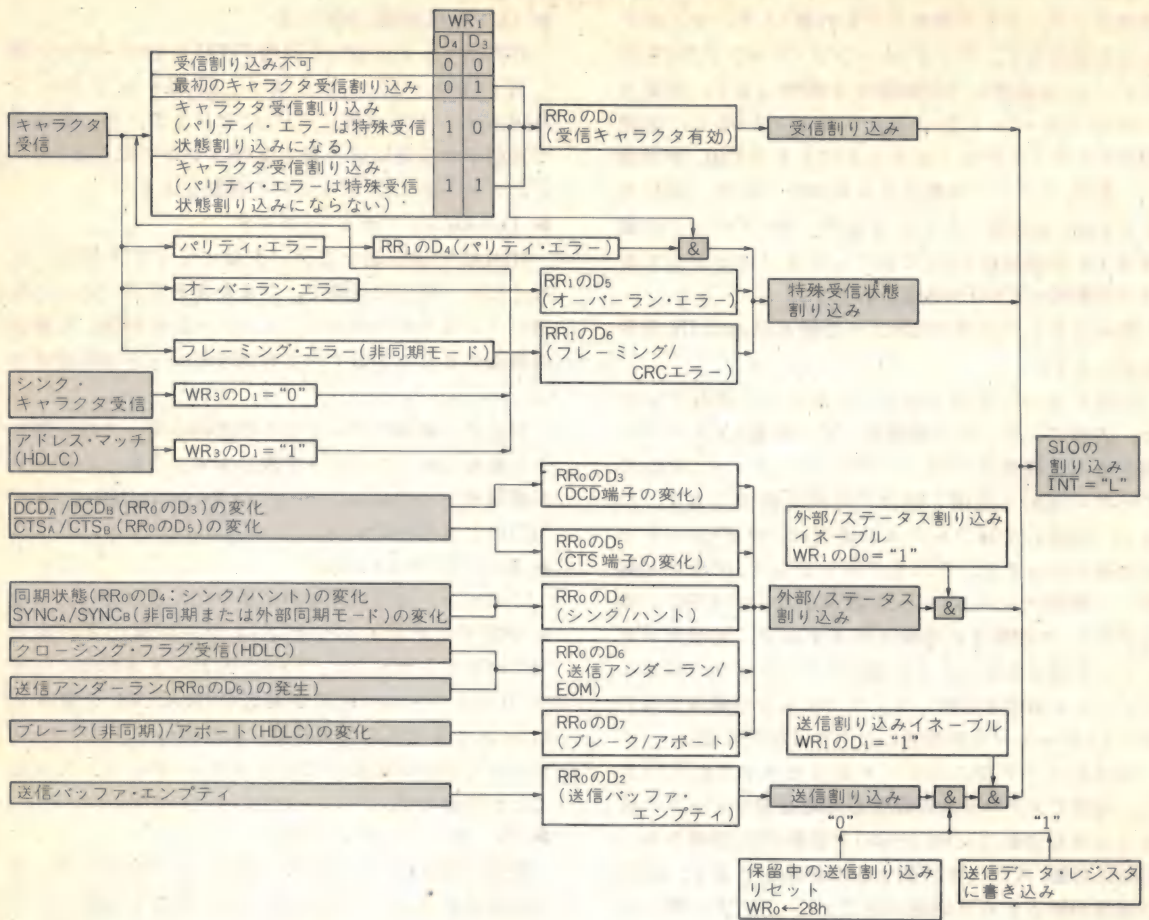
非同期モードでは受信データにフレーミング・エラー (ストップ・ビットであるべきビットがデータであった) が発生したことを、同期モードおよび HDLC モードではそれまでの CRC チェックに誤りがあったことを表します。

いったんセットされるとエラー・リセット・コマンドが実行されるまでそのままです。

#### ▶ D<sub>7</sub>: エンド・オブ・フレーム (EOM)

このビットは HDLC モードのみで使用されます。これはフレームの受信が正常に (CRC エラーなしでクロージング・フラグを受信) 終了したときにセットされ、次のフレームの最初のデータでリセットされます。

〈図6〉 Z80SIOの割り込みシステム



## ● 読み出しレジスタ 2 (RR<sub>2</sub>)

このレジスタはチャンネル B にしかありません。このレジスタは CPU に対して発生した割り込みのベクタを保持します。これは CPU が Z80 以外の場合や割り込みモードが 2 以外の場合に、バスにベクタは送れないが、割り込み原因の解析を簡単に行いたい場合に読むものです。よほどの理由がない限り、ステータス・アフェクト・ベクタ機能により割り込みベクタを変えて要因を判定するのが一般的で、このレジスタを読む込むことはしないでしょ。

## SIO のデータ送受信割り込み

Z80SIO は他の CTC や PIO に比較して非常に高性能で、いろいろな割り込みを発生させることができます。ここで Z80SIO の割り込みについて図 6 にまとめてみます。

割り込みには大きく分けて、受信割り込み、送信割り込み、外部/ステータス割り込み、特殊受信状態割

り込みの四つがあります。

またこれらの四つの割り込みは、ステータス・アフェクト・ベクタ (WR<sub>1</sub> の D<sub>2</sub>) がセットされていれば、チャンネルごとにそれぞれ独立した割り込みベクタを発生するので、割り込みルーチンの先頭での割り込み要因判別ルーチンが不要になります。

## ● 受信割り込みのいろいろ

図 7 をみてください。一般に非同期のシリアル通信では、キャラクタとキャラクタの間隔は任意です。いつ受信されるかわからないデータをポーリングで待つのは不利です。受信割り込みとはこのように、非同期に送られてくるデータの受信に最適で、データが送られてくる間は CPU は別のことを処理できます。

SIO の受信割り込みは 3 通り設定できます。一般的にはキャラクタ受信時の割り込み (パリティ・エラーは特殊受信状態になる) を使用します。

Z80SIO ではパリティ・エラーの扱いが少し特殊で、キャラクタ受信割り込みに入れることも、特殊受信状態割り込みに入れることもできます。当然キャラクタ



受信割り込みに入れると、パリティ・エラーで特殊受信状態割り込みは発生しません。ただしパリティ・エラーのビット(RR<sub>1</sub>のD<sub>4</sub>)は、どちらに設定されていても、パリティ・エラーが発生すると“1”になります。

### ● 最初のキャラクタのみ割り込み

“最初のキャラクタの割り込み”と聞いても何のことかよくわからない方もいるでしょう。図7(b)のように、データがキャラクタ単位ではなく、あるブロックごとに固まって送られてくる場合があります。このときは最初のキャラクタのみ、割り込みで受信し、後のデータはポーリングで受信するという方法もあります。

しかし最初のキャラクタ受信割り込みで次のブロック・データをポーリングする(RR<sub>0</sub>のD<sub>0</sub>が“1”になるのを待つ)方法では、その間に別の割り込み(タイマ割り込みやSIOの他チャンネルの割り込みなど)の処理ができません。筆者の考えとしてはあまり有効な受信割り込みとは思えません。

普通にキャラクタ受信割り込みを使うのが一般的だと思います。

### ● 送信割り込み

受信割り込みは、相手が送ってきたデータを受信したときに発生する割り込みですが、送信割り込みとは、自分の送信バッファに空きができたとき発生する割り込みです。つまり、例えば100バイト送信したら終わりというときでも、100バイト目を送信し終えた後で割り込みが発生します。もう送信するデータがないのにもかかわらずです。

保留中の送信割り込みリセット(WR<sub>0</sub>に28hを書き込む)とはこのように、送信割り込みが発生したにもかかわらず、送信すべきデータがなくなったときに行います。

このコマンドを実行せずに、しかも送信データを何も書き込むことなく割り込みルーチンから復帰(RETI命令を実行)すると、すぐにまた送信割り込みが発生し、結果的にメイン・ルーチンが動けなくなります。

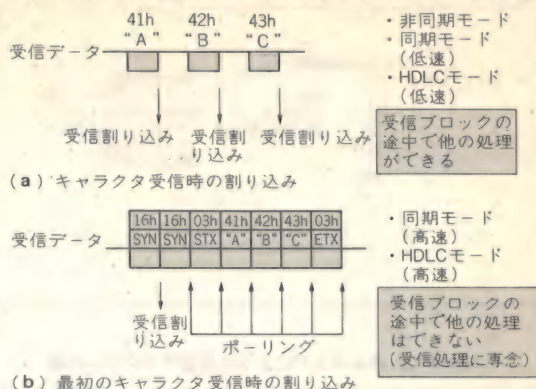
次に送信したいデータが発生した場合には、そのまま送信したいデータをデータ・ポートに書き込むだけでOKです。そのデータが送信されて、送信バッファがエンプティになると、また再び送信割り込みを発生するようになります。

### ● 特殊受信状態/外部/ステータス割り込み

受信エラーのときは特殊受信状態割り込みが発生しますが、何のエラーかまで判別したいときは、RR<sub>1</sub>を読み出して判別します。

同じように、外部/ステータス割り込みについても、RR<sub>0</sub>を読み出せば詳しい割り込み発生の原因がわかります。

〈図7〉キャラクタ受信割り込みと最初のキャラクタ受信割り込み



またこれらの割り込みはWR<sub>0</sub>に外部/ステータス割り込みリセット、もしくはエラー・リセット・コマンドを書き込まないと、再度割り込みが発生しないので注意してください。

## SIO のプログラミング

### ● SIOの初期化

SIOのレジスタ初期化には、リスト1のようにOTIR命令(Bレジスタが0になるまで、HLレジスタが示すアドレスのデータをCレジスタが示すポートに出力し、HLレジスタを+1する)を使うのが便利です。

リストの最初のDI命令は、初期化中に割り込みが発生しないようにするための安全策です。

実際のパラメータは、リスト1のようにレジスタ・ポインタ・コマンドとそのレジスタへのパラメータを順番に並べます。

### ● 3線式シリアル伝送

SIOをモデムに接続するのではなく単純なシリアル伝送(送受信)に使用するならば、線が3本(TxD, RxD, GND)の3線式シリアル伝送方式で十分です。

この場合、SIOのCTS/DCD端子はオート・イネーブルが設定されても動作するように、GNDに接続しておくのが安全です。

どうしても入力端子を汎用の入力ポートとして使いたいときは、オート・イネーブルに設定してはなりません。

図8に回路例を、リスト2にプログラム例を示します。

### ● ポーリングによるデータ送受信

データの送信するにはSIOを初期化しイネーブル・トランスミッタをセットします。そしてまず最初にRR<sub>0</sub>のビット2(送信バッファ・エンプティ)を調べて“0”ならば“1”になるまで待ちます。

# 〈リスト 1〉 SIO のチャンネル初期化プログラム例

```

; SIOチャンネルAの初期化
SIO_A_INITIAL: DI
               LD B, 9
               LD C, SIO_AC
               LD HL, SIO_A_PARAM
               OTIR
               EI
               RET
               ;割り込み禁止
               ;パラメータのバイト数
               ;初期化するSIOアドレス
               ;パラメータのアドレス
               ;SIOの初期化
               ;割り込み許可

```

```

; SIOのパラメータ
SIO_A_PARAM: DB 18H
              DB 1
              DB 18H
              DB 4
              DB 47H
              DB 3
              DB 41H
              DB 5
              DB 0AAH
              ;チャンネル・リセット
              ;WR1 送信割り込みなし
              ;キャラクタ受信時の割り込み
              ;WR4 CLOCK #16
              ;STOP bit 1 偶数パリティ
              ;WR3 受信キャラクタ 7bit
              ;イネーブル・レシーバ
              ;WR5 送信キャラクタ 7bit
              ;イネーブル・トランスミッタ

```

## 〈リスト 2〉 ボーリングによる送受信プログラムの例

### ; SIOボーリング送受信プログラム (3 線式)

```

; I/Oアドレス定義
SIO_AD EQU 18H
SIO_AC EQU 19H
SIO_BD EQU 1AH
SIO_BC EQU 1BH
CTC_CO EQU 10H

; SIOチャンネルAデータ
; SIOチャンネルAコントロール
; SIOチャンネルBデータ
; SIOチャンネルBコントロール
; CTCチャンネル0

ORG 0
LD SP, 0000H
CALL CTC_0_INITIAL
CALL SIO_A_INITIAL
JP MAIN

; CTCを使った送受信クロックの設定 (例)
CTC_0_INITIAL: LD A, 47H
               OUT (CTC_CO), A
               LD A, 16
               OUT (CTC_CO), A
               RET

; SIOチャンネルAの初期化
SIO_A_INITIAL: DI
               LD B, 7
               LD C, SIO_AC
               LD HL, SIO_A_PARAM
               OTIR
               EI
               RET
               ;割り込み禁止
               ;パラメータのバイト数
               ;初期化するSIOアドレス
               ;パラメータのアドレス
               ;SIOの初期化
               ;割り込み許可

; SIOのパラメータ
SIO_A_PARAM: DB 18H
              DB 4
              DB 47H
              DB 3
              DB 41H
              DB 5
              DB 0AAH
              ;チャンネル・リセット
              ;WR4 CLOCK #16
              ;STOP bit 1 偶数パリティ
              ;WR3 受信キャラクタ 7bit
              ;イネーブル・レシーバ
              ;WR5 送信キャラクタ 7bit
              ;イネーブル・トランスミッタ

; 1キャラクタ送信 (ボーリング)
SEND_BYTE: PUSH AF
SEND_BYTE_1: IN A, (SIO_AC)
              AND 4
              JR Z, SEND_BYTE_1
              POP AF
              OUT (SIO_AD), A
              RET
              ;Aレジスタ/フラグの保存
              ;RR0をAレジスタに読み込み
              ;送信バッファ・エンプティ
              ;"0"の場合は待つ
              ;Aレジスタ/フラグを復元
              ;送信データを書き込み
              ;サブルーチンからの復帰

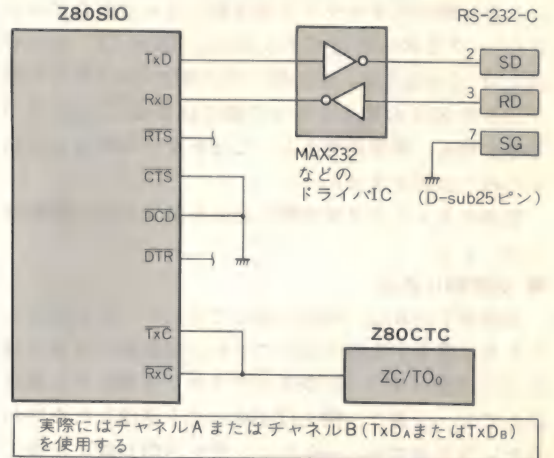
; 1キャラクタ受信 (ボーリング)
RECEIVE_BYTE: IN A, (SIO_AC)
               AND 1
               JR Z, RECEIVE_BYTE
               IN A, (SIO_AD)
               RET
               ;RR0をAレジスタに読み込み
               ;受信データ有効
               ;"0"の場合は待つ
               ;受信データを読み込み
               ;サブルーチンからの復帰

; RR1をAレジスタに読み込み
INPUT_RR1: DI
            LD A, 1
            OUT (SIO_AC), A
            IN A, (SIO_AC)
            EI
            RET
            ;割り込み禁止
            ;ポイント・コマンド
            ;コマンド実行 (RR1選択)
            ;エラー情報 (RR1)の読み込み
            ;割り込み許可

; コマンド (WR0) 実行の例 (エラー・リセット)
RESET_ERROR: LD A, 30H
              OUT (SIO_AC), A
              RET
              ;エラー・リセット・コマンド
              ;コマンド実行
              ;サブルーチンからの復帰

```

〈図 8〉 3 線式の RS-232-C インターフェース回路例



送信バッファ・エンプティならば SIO のデータ・ポートに送信したいデータを書き込むと送信されます。ただしオート・イネーブルがセットされているときは、CTS 端子が L レベルにならないと送信されません。

データの受信をするには SIO を初期化しイネーブル・レシーバをセットしてから、RR<sub>0</sub>のビット 0 (受信データ有効)を調べて、“0” ならば “1” になるのを待ちます。

受信データ有効ならば SIO のデータ・ポートから受信データを読み込みます。またオート・イネーブルがセットされているときは、DCD 端子が L レベルになっていないとデータ受信ができません。

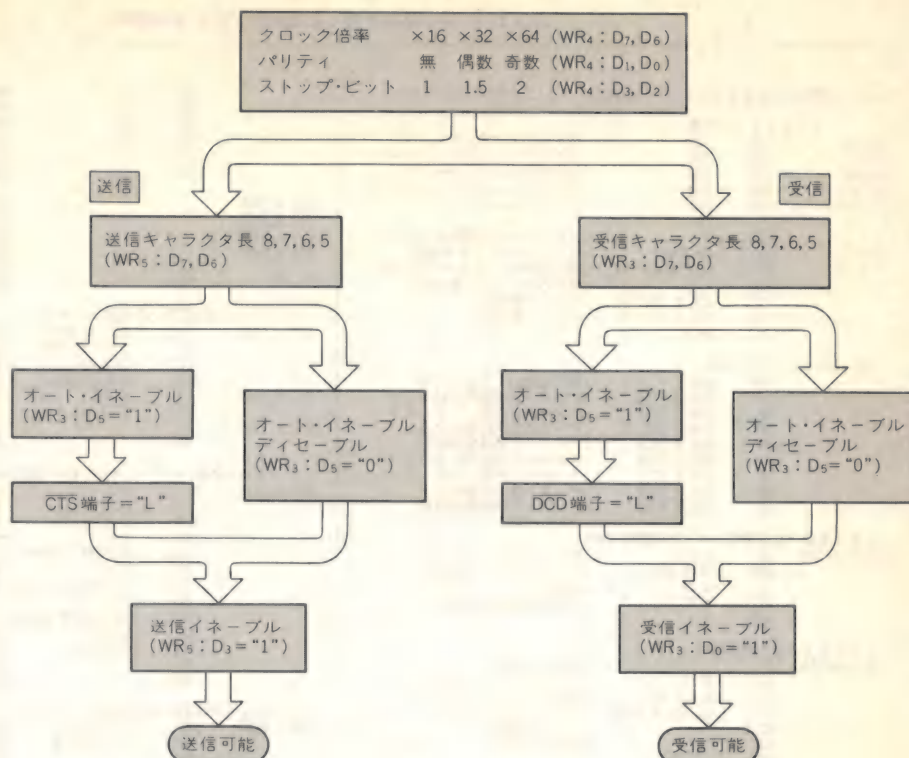
図 9 に、いちばん基本的な SIO の送受信設定のチェック・ポイントを示します。少なくとも非同期通信 3 線式ボーリング送受信であれば、これらの項目をきちんと設定できれば、データの送受信ができるはずです。

### ● チャンネル A の DTR を制御する

リスト 3 に例を示します。WR<sub>0</sub>以外のレジスタの設定を変える場合は、割り込みルーチン処理により別のレジスタに書き込まれるのを防ぐためにまず割り込みを禁止します。



〈図9〉  
SIOの基本設定  
チェック・ポイント



そしてコントロール・ポートに書き込みたいレジスタ・ポインタを出力することにより書き込みレジスタを選択し、次に設定したいデータを同じくコントロール・ポートに出力します。

またすでに述べたように、SIOは書き込みレジスタの現在の状態を読み出すことができません。他のビットの状態がわからず、またその状態を変えずに特定のビットだけをON/OFFしたいときは、メモリ上に書き込んだデータを保存しておき、そのデータに対してOR命令やAND命令で特定のビットをセットしたり、リセットしたデータを書き込みます。

もちろん、初期化した後にも変更していないときなど、この例でいえばDTR以外のビットの状態がわかりきっているときなどは、ワークエリアに保存などしないで、直接書き込んでもよいでしょう。

### ● 割り込みを使ったデータ送受信

リスト4に割り込みを使ったデータ送受信プログラムを、図10に回路例を示します。割り込みを使う場合は設定が必要なレジスタの数も増えます。

送信や受信は、割り込み発生により直接それぞれのルーチンにジャンプするので、ポーリングのときのようなRRoなどのビットの状態をチェックする必要がなく、ポーリングのときより簡単になります。

### ● 割り込みとHDLCモードでの初期化

リスト5に割り込みを使ったHDLCモードでのプ

〈リスト3〉DTR端子をコントロールするプログラム例

```

; チャネルAのDTRをLレベル (E R信号をオン)
CH_A_ER_ON:  DI      AF      ; 割り込み禁止
              PUSH    AF      ; Aレジスタ/フラグを保存
              LD       A, 5    ; レジスタ・ポインタ・コマンド
              OUT      (SIO_AC), A ; WR5を指定
              LD       A, (SIOA_WR5) ; WR5保存データ読み出し
              OR       80H     ; bit7 ON (DTR = "L")
              OUT      (SIO_AC), A ; WR5を書き換え
              LD       A, (SIOA_WR5), A ; WR5を保存
              POP      AF      ; Aレジスタ/フラグを復元
              EI          ; 割り込み許可
              RET            ; サブルーチンからの復帰

; チャネルAのDTRをHレベル (E R信号をオフ)
CH_A_ER_OFF: DI      AF      ; 割り込み禁止
              PUSH    AF      ; Aレジスタ/フラグを保存
              LD       A, 5    ; レジスタ・ポインタ・コマンド
              OUT      (SIO_AC), A ; WR5を指定
              LD       A, (SIOA_WR5) ; WR5保存データ読み出し
              AND      07FH     ; bit7 OFF (DTR = "H")
              OUT      (SIO_AC), A ; WR5を書き換え
              LD       A, (SIOA_WR5), A ; WR5を保存
              POP      AF      ; Aレジスタ/フラグを復元
              EI          ; 割り込み許可
              RET            ; サブルーチンからの復帰

```

ログラムを、図11にインターフェースの回路例を示します。

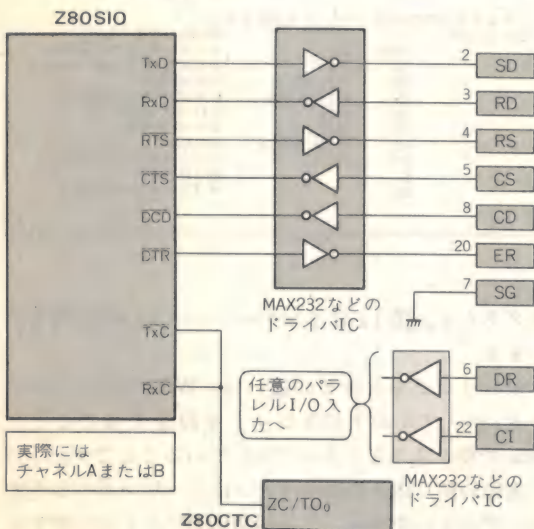
ポーリングのときとの違いは、WR<sub>2</sub>に割り込みベクタ・テーブルの下位8ビットを設定することと、WR<sub>1</sub>で割り込みをイネーブルしていることです。

実際の割り込み処理ルーチンはポーリングのときと同じでかまいません。この初期化例でとくに注意することは、チャネルAしか使っていないにもかかわらず

〈リスト4〉 割り込みを使った送受信プログラムの例

: SIO割り込み送受信プログラム (7線式オート・イネーブル・モード)		
: I/Oアドレス定義		
SIO_AD	EQU	18H
SIO_AC	EQU	19H
SIO_BD	EQU	1AH
SIO_BC	EQU	1BH
CTC_CO	EQU	10H
: SIOチャネルAデータ		
: SIOチャネルAコントロール		
: SIOチャネルBデータ		
: SIOチャネルBコントロール		
: CTCチャネル0		
ORG	0	: プログラム実行開始アドレス
LD	SP, 0000H	: スタック・ポインタの設定
LD	A, HIGH VECTOR	: 割り込みベクタ・テーブル上位
LD	I, A	: 割り込みレジスタを設定
CALL	CTC_0_INITIAL	: CTCの初期化
CALL	SIO_A_INITIAL	: SIOの初期化
JP	MAIN	: メイン・ルーチンへジャンプ
: 割り込みベクタ・テーブル		
VECTOR:	ORG	80H
DW	SIO_B_TXBE	: チャネルB送信割り込み
DW	SIO_B_ESTI	: チャネルB外部/ステータス
DW	SIO_B_RXCA	: チャネルB受信割り込み
DW	SIO_B_RXSP	: チャネルB特殊受信状態
DW	SIO_A_TXBE	: チャネルA送信割り込み
DW	SIO_A_ESTI	: チャネルA外部/ステータス
DW	SIO_A_RXCA	: チャネルA受信割り込み
DW	SIO_A_RXSP	: チャネルA特殊受信状態
: CTCを使った送受信クロックの設定 (例)		
CTC_0_INITIAL:	LD	A, 47H
OUT	(CTC_CO), A	: CTCのモード・コマンド
LD	A, 16	: モード・コマンドを出力
OUT	(CTC_CO), A	: 分周比
RET		: 分周比を出力 (設定)
: SIOチャネルAの初期化		
SIO_A_INITIAL:	DI	: 割り込み禁止
LD	B, 11	: パラメータのバイト数
LD	C, SIO_AC	: 初期化するSIOアドレス
LD	HL, SIO_A_PARAM	: パラメータのアドレス
OTIR		: SIOの初期化
EI		: 割り込み許可
RET		
: SIOのパラメータ		
SIO_A_PARAM:	DB	18H
		: チャネル・リセット
	DB	2
		: WR2
	DB	LOW VECTOR
		: 割り込みベクタ設定
	DB	1
		: WR1 受信割り込み
	DB	1FH
		: 割り込みイネーブル
: チャネルA送信割り込み処理ルーチン		
SIO_A_TXBE:	CALL	GET_TX_BYTE
	JR	Z, SIO_A_TXBE_1
	OUT	(SIO_AD), A
	JR	SIO_A_TXBE_2
SIO_A_TXBE_1:	LD	A, 28H
	OUT	(SIO_AC), A
SIO_A_TXBE_2:	EI	
	RETI	
: チャネルA外部/ステータス割り込み処理ルーチン		
SIO_A_ESTI:	LD	A, 1
	OUT	(SIO_AC), A
	IN	A, (SIO_AC)
	BIT	4, A
	CALL	NZ, PARITY_ERROR
	BIT	5, A
	CALL	NZ, OVERRUN_ERROR
	BIT	6, A
	CALL	NZ, FRAMING_ERROR
	LD	A, 10H
	OUT	(SIO_AC), A
	EI	
	RETI	
: チャネルA受信割り込み処理ルーチン		
SIO_A_RXCA:	IN	A, (SIO_AD)
	CALL	PUT_RX_BYTE
	EI	
	RETI	
: チャネルA特殊受信状態割り込み処理ルーチン		
SIO_A_RXSP:	LD	A, 30H
	OUT	(SIO_AC), A
	EI	
	RETI	

〈図10〉 RTSやCTS端子も使用したRS-232-Cインターフェースの回路例



ず、チャネルBのWR<sub>1</sub>とWR<sub>2</sub>にも書き込みを行っていることです。

この理由は、SIOには割り込みベクタ・レジスタとステータス・アフェクト・ベクタの設定ビットはチャネルBにしか存在しないため、たとえチャネルAしか使わない場合でも、割り込みを使う場合はチャネルBも初期化しなければならないことです。

### ● 最後に

SIOを使う際の基礎からレジスタの詳細、プログラミングについて解説しましたが、SIOは果てしなく高性能であるというのが実感です。モードによるプログラミングの違い、特に同期モードとHDLCモードについては機能が複雑で、ある順番にコマンドを実行し、レジスタを書き換えなければ動作しません。これらのプログラムは限られた紙面ではなかなかお伝えできません。そこでディスク頒布サービスでこれを行うこととしました。

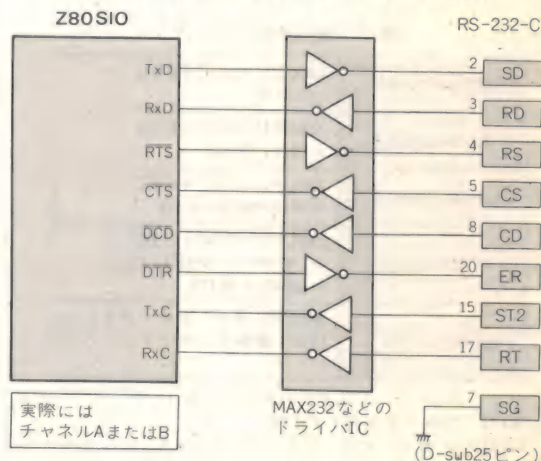
このディスクには、SIOの非同期/同期/HDLCモードで実際にそのまま動作するプログラムと、



〈リスト5〉 割り込みを使った HDLC フレーム送受信プログラムの例

HDLC 送受信プログラム		
SIOAD EQU 0	: SIO チャネルAデータ・アドレス	
SIOAC EQU 1	: SIO チャネルAコントロール・アドレス	
SIOBD EQU 2	: SIO チャネルBデータ・アドレス	
SIOBC EQU 3	: SIO チャネルBコントロール・アドレス	
ORG 0	: ROM開始アドレス	
DI	: 割り込み禁止	
LD SP, 0	: スタック・ポインタの設定	
IM 2	: 割り込みモード2	
LD A, HIGH(INTVT)	: 割り込みベクタ上位8ビット	
LD A, 1	: 割り込みレジスタの設定	
JR INISIO	: SIOの初期化へジャンプ	
: 割り込みベクタ・テーブル		
INTVT: DEFW 0, 0, 0, 0		
DEFW TXBEI	: 送信バッファ・エンプティ	
DEFW EXSTI	: 外部/ステータス割り込み	
DEFW RXCAI	: 受信キャラクタ有効	
DEFW SPRXI	: 特殊受信状態割り込み	
: SIOの初期化		
INISIO: LD HL, SIOAP	: チャネルAパラメータ・テーブル	
LD B, 13	: チャネルAパラメータの個数	
LD C, SIOAC	: チャネルAコントロール・アドレス	
OTIR	: SIOチャネルAの初期化	
LD HL, SIOBP	: チャネルBパラメータ・テーブル	
LD B, 5	: チャネルBパラメータの個数	
LD C, SIOBC	: チャネルBコントロール・アドレス	
OTIR	: SIOチャネルBの初期化	
JR INIWRK	: ワーク・エリアの初期化へ	
: SIOチャネルAパラメータ		
SIOAP: DEFB 98H	: レジスタ・アドレス・レジスタ・アドレス	
DEFB 4	: レジスタ・アドレス・レジスタ・アドレス	
DEFB 20H	: WR4=90H, HDLC=1	
DEFB 1	: レジスタ・アドレス・レジスタ・アドレス	
DEFB 13H	: WR1=13H, TX/RX/STATUS割込	
DEFB 3	: レジスタ・アドレス・レジスタ・アドレス	
DEFB 0DDH	: WR3=RX 8BITS, HUNT, RX CRC, #4	
DEFB 6	: レジスタ・アドレス・レジスタ・アドレス	
DEFB 1	: WR6=サーチ・アドレス=1	
DEFB 7	: レジスタ・アドレス・レジスタ・アドレス	
DEFB 7EH	: WR7=7EH, シーケンス・パターン	
DEFB 5	: レジスタ・アドレス・レジスタ・アドレス	
DEFB 0EBH	: WR5=DTR, CCITT, RTS, TX CRC	
: SIOチャネルBパラメータ		
SIOBP: DEFB 18H	: レジスタ・アドレス・レジスタ・アドレス	
DEFB 1	: レジスタ・アドレス・レジスタ・アドレス	
DEFB 4	: ステータス・レジスタ・アドレス・レジスタ・アドレス	
DEFB 2	: レジスタ・アドレス・レジスタ・アドレス	
DEFB LOW(INTVT)	: 割り込みベクタ・テーブル下位8bit	
: ワーク・エリアの初期化		
INIWRK: TXCF, TXDL, RXCF, RXDLを0に		
: TXDPを送信・受信中のレジスタに設定		
EI	: 割り込みイネーブル	
: メイン・ルーチン		
: ~ 途中省略 ~		
TXFRME: LD HL, TXBD	: 送信データ・バッファ・アドレス	
LD (TXDP), HL	: 送信データ・ポインタの設定	
LD (HL), A	: コントローラ・レジスタを送信バッファへ書き込み	
XOR A	: Aレジスタをクリア	
LD (TXCF), A	: 送信完了フラグをリセット	
LD A, 10	: 送信するデータのバイト数	
LD (TXDL), A	: STORE TRANSMIT DATA LENGTH	
LD A, 1	: アドレス・フィールド	
OUT (SIOAD), A	: 送信開始(送信割込イネーブル)	
LD A, 0COH	: 送信アンダーランRESETコマンド	
OUT (SIOAC), A	: 送信アンダーランのリセット	
: ~ 省略 ~		
: 送信バッファ・エンプティ割り込み処理ルーチン		
TXBEI: PUSH AF	: Aレジスタとフラグの退避	
PUSH HL	: HLレジスタの退避	
LD A, (TXDL)	: 残りの送信データのバイト数	
OR A	: ゼロかどうかの判定	
JR Z, TXEND	: 残りの送信データがない	
DEC A	: 残りの送信データのバイト数を-1	
: 外部/ステータス割り込み処理ルーチン		
EXSTI: PUSH AF	: Aレジスタとフラグの退避	
PUSH HL	: HLレジスタを退避	
LD A, (RXCF)	: RROを読み出し	
OR A	: RROを保存	
JR NZ, RXEND	: 外部/ステータス割り込みリセットコマンド	
LD A, (RXDL)	: Aレジスタとフラグの復旧	
INC A	: 割り込みイネーブル	
JR Z, RXEND	: 割り込みからの復帰	
LD (RXDL), A		
IN A, (SIOAD)		
LD HL, (RXDP)		
INC HL		
LD (HL), A		
INC HL		
LD (HL), A		
JR RAXEXIT		
RXEND: IN A, (SIOAD)		
RXEXIT: POP HL		
POP AF		
EI		
RETI		
: 特殊受信状態割り込み処理ルーチン		
SPRXI: PUSH AF	: Aレジスタとフラグの退避	
LD A, 1	: RRIポインタ・コマンド	
OUT (SIOAC), A	: RRIを選択	
IN A, (SIOAC)	: RRIを読み込み(入力)	
LD (RRI), A	: RRIを保存	
BIT 7, A	: フレームの終わるか?	
JR Z, NOPAC	: 終わりでなければジャンプ	
LD A, (RXDL)	: 受信フレームのバイト数を讀込	
DEC A	: FCSが1バイト受信されるので無視	
LD (RXDL), A	: 実際のフレームの内容を保存	
LD A, 1	: 受信完了フラグをセット	
(RXCF), A	: エラー・リセット・コマンド	
A, 30H	: エラー・リセット(割込許可)	
OUT (SIOAC), A	: Aレジスタとフラグを復旧	
POP AF	: 割り込みイネーブル	
EI	: 割り込みからの復帰	
RETI		

〈図11〉 同期モード時のインターフェース回路例



Z80SIOの機能をふんだんに使ったシリアル・ライン・モニタのプログラム、フレーム・レベルのHDLC送受信プログラムのサンプルが収められています。ご活用ください。

●参考文献●

- (1) VOLUME 1 DATABOOK, MICROPROCESSORS AND PERIPHERALS, ZILOG.
- (2) Z80 Family User's Manual, ZILOG.

東芝TMPZ84C0xx/ザイログZ84Cxx/川崎製鉄KL5C8012の使い方

## 周辺組み込み & Z80互換高速CPUの活用

高見 豊/菅原尚伸/菊地恭徳/北 孝

実際にマイコン・システムを設計する場合は、CPUとメモリだけでは役に立たず、必ずCTCやPIOなどのI/Oが必要になります。そこで、CPUと使用頻度の非常に高い周辺LSIを一つのICの中に入れてしま

った、ワンチップ・マイコンというものがいくつか出ています。

ここではZ80CPUを核にして周辺LSIを組み込んだCPUについて取り上げてみます。

### TMPZ84C0xx シリーズ

もともとTMPZ84Cシリーズは、東芝製のZ80ファミリを示す型番でしたが、最近ではTMPZ84C015に代表される、Z80CPUを核にZ80ファミリを内蔵したシリーズが有名です。

表1にTMPZ84Cシリーズを示します。周辺LSIの組み合わせで、いくつかシリーズがあります。ここではよく使われるC015、C013、C011について解説します。

#### TMPZ84C015

##### ● ピン配置、ピン名称

ピン配置を図1に示します。基本的にZ80CPUおよびCTCやPIO、SIOと同じ名前のピンは、いままで説明してきた単体のCPUや周辺LSIの動作とまったく同じです。ただしTMPZ84C015で拡張された機

能などのために、特有の信号ピンがいくつかあります(表2)。

TMPZ84C015ではCGC(クロック・ジェネレータ・コントローラ)を内蔵しているの、XTAL<sub>1</sub>とXTAL<sub>2</sub>間に図2のように水晶振動子とコンデンサを接続するだけで、クロックを発生させることができます。ここに接続する水晶発振子は、動作させたいクロック周波数の2倍の周波数のものを接続します。

IEOとIEIは、Z80ファミリLSIの割り込み優先順位の制御線です。内蔵のI/O以外に、さらに外付けにZ80ファミリを接続し、割り込みを使用するときには接続が必要になります。外付けにI/Oを増設しないときや、増設したとしてもZ80ファミリではないときは、IEIをプルアップしておきます。IEOは解放でかまいません。

CLKINは外部クロック入力端子で、内蔵CGCを使用せずに外部からクロックを入力するときに使用します。単体Z80CPUのCLK端子と考えてください。通常はCLKOUTに接続します。

CLKOUTは内蔵CGCで2分周されたシステム・クロックの出力端子です。さらにTMPZ84C015には、内蔵のCPUを切り離すためのEV端子があります。これはICEなどのツールを使ってシステムを開発するときには使用する端子で、通常はプルダウンしておきます。またICT端子はICテスト用の端子なので解放のままにしておきます。

##### ● Z80ファミリLSIとの違い

もう一つ注意すべき点は、Z80CTCではパッケージのピン数の関係で省略されていたチャンネル3のZC/TO<sub>3</sub>の出力ピンがある点、そしてZ80SIOのボンディ

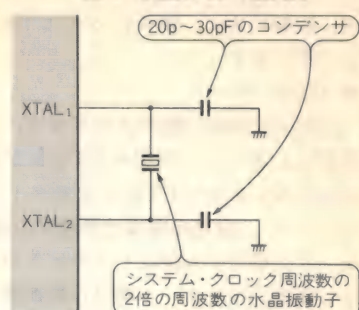
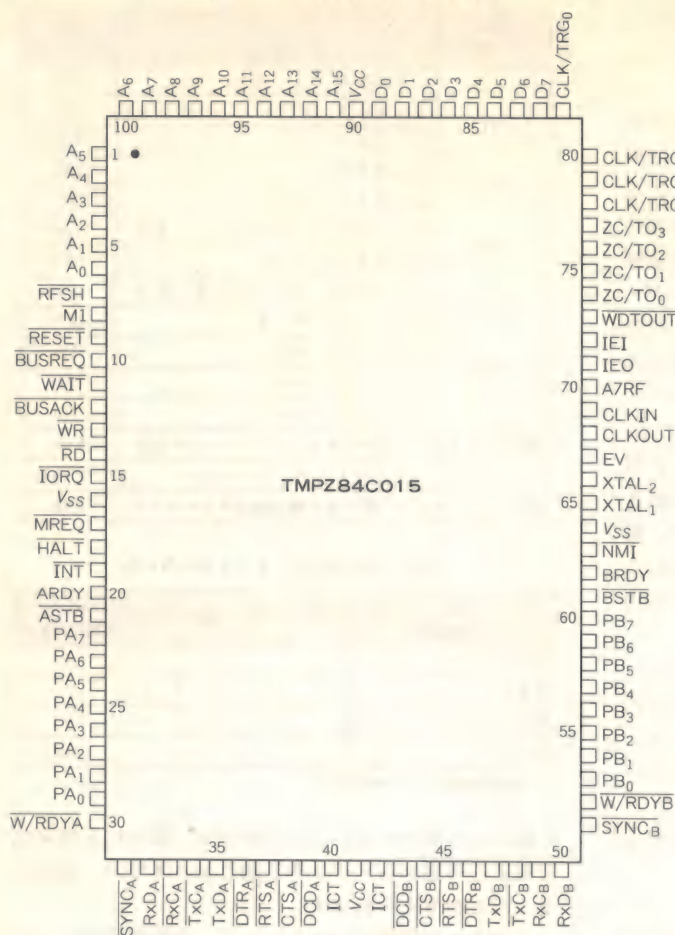
〈表1〉TMPZ84Cシリーズ

TMPZ84C011AF	Z80CPU+CGC+CTC +I/Oポート
TMPZ84C013AF	Z80CPU+CGC+CTC +SIO+WDT
TMPZ84C015AF	Z80CPU+CGC+CTC +PIO+SIO+WDT
TMPZ84C810AF	Z80CPU+CGC+CTC +I/Oポート+SIO+DMA +MMU+WDT
TMPZ84C710A	Z80CPUをコントローラとした ISDN基本インターフェース用 チップ

CGC：クロック・ジェネレータ・コントローラ、  
WDT：ウォッチドグ・タイマ、  
MMU：メモリ・マネージメント・ユニット



〈図 2〉 水晶発振子の接続例



ピン名称	入出力	機能
XTAL <sub>1</sub>	入力	水晶発振子用接続端子
XTAL <sub>2</sub>	出力	
IEO	出力	割り込みイネーブル出力
IEI	入力	割り込みイネーブル入力
CLKIN	入力	外部クロック入力
CLKOUT	出力	クロック出力
EV(I)	入力	エバリュエータ用信号
A7RF	出力	補助アドレス・ビット
WDTOUT	出力	ウォッチドッグ・タイマ出力
ICT	出力	テスト用端子

TMPZ84Cでは、HALT命令実行時のクロックに関する動作を4通り設定することが可能です。この

HALT モードの設定は CPU の動作や消費電力に大きく影響します。表 4 にスタンバイ・モード設定について示します。

#### ▶ IDLE1 モード

内部の発振器は動作させますが、外部へのクロック供給はしません。内部の CTC などにもクロックは供給されないの、CTC からのタイマ割り込みによるリスタート(設定時間後に処理を再開する)はできません。この場合は外部にリスタートのための回路を必要とします。リスタート時には発振器のウォーミング・アップが必要ないので、素早く処理を再開させることができますが、発振器が動作しているために、数 mA の電流を消費します。

#### ▶ IDLE2 モード

内部発振器を動作させ、さらに外部へのクロック供給をします。内部の CTC などにもクロックは供給されるので、CTC からの割り込みによるリスタート(設定時間後に処理を再開する)ができます。しかし、CTC が動作しているために、消費電流は通常の半分程度にしかありません。

#### ▶ STOP モード

内部動作をすべて停止するので、消費電流はほとんどゼロになります。しかしリスタート時にはウォーミング・アップを行うので、処理の再開には  $2^{14}$  クロック・サイクルの時間がかかります。

なお、外部クロックを使用すると、HALT 命令を実行しても STOP モードにはなりません。

#### ▶ RUN モード

HALT 命令を実行してもスタンバイ・モードにはなりません。したがって消費電流も通常の動作時と同じになります。つまり Z80 と同様に、内部的には NOP 命令の実行を繰り返すモードです。

### ● スタンバイ・モードの設定

システム設定に関する重要な設定なので、簡単には変更できないようなしかけがされています。まずホールト・モード設定レジスタ(アドレス F1h)に対して DBh を書き込みます。それからホールト・モード・コントロール・レジスタ(アドレス F0h)のビット 4, 3 で、ホールト命令実行時のスタンバイ・モードを設定します。またこのレジスタの上位ビットは、後述するウォッチドグ・タイマのレジスタになっているので、ウォッチドグ・タイマでの設定も考えてデータを書き込みます。

### ● ウォッチドグ・タイマとは

マイコンの処理は、いくつかある処理を決められた順序で繰り返すという動作が大半をしめていると思われます。

また本来ハードウェアやソフトウェアが正常であれば、暴走などという状態にはならないはずですが、しか

〈表 3〉 TMPZ84C015 内蔵 I/O のポート・アドレス

内蔵 I/O デバイス	チャンネル	I/O アドレス
CTC	チャンネル 0	10h
	チャンネル 1	11h
	チャンネル 2	12h
	チャンネル 3	13h
SIO	チャンネル A データ	18h
	チャンネル A コマンド/ステータス	19h
	チャンネル B データ	1Ah
	チャンネル B コマンド/ステータス	1Bh
PIO	チャンネル A データ	1Ch
	チャンネル A コマンド	1Dh
	チャンネル B データ	1Eh
	チャンネル B コマンド	1Fh
ウォッチドグ・タイマ/スタンバイ・モード設定		F0h
ウォッチドグ・タイマ・コマンド・レジスタ		F1h
デジジ・チェーン割り込み優先順位設定レジスタ		F4h

〈表 4〉 スタンバイ・モードのいろいろ

モード	CGC 発振器	CPU	CTC	PIO	SIO	WDT	CLK OUT 端子
IDLE1	○	×	×	×	×	×	×
IDLE2	○	×	○	×	×	×	○
STOP	×	×	×	×	×	×	×
RUN	○	○	○	○	○	○	○

○：動作継続，×：動作停止

し組み込み用マイコンなどの場合は、置かれる環境などにより外部からノイズや予期しない信号で、誤動作することが考えられます。

このような状態になったときに、自動的にリセットをかけられるシステムがあると、ふたたび決められた処理を繰り返す動作に復帰でき便利です。

これを実現するのがウォッチドグ・タイマです。一定時間以内にプログラムがカウンタの値をクリアしないと、リセットに接続された出力がアクティブになり、システムをリセットするというものです。よってソフトウェアで、その一定時間以内にカウンタをクリアするようにしなければなりません。

### ● ウォッチドグ・タイマの設定

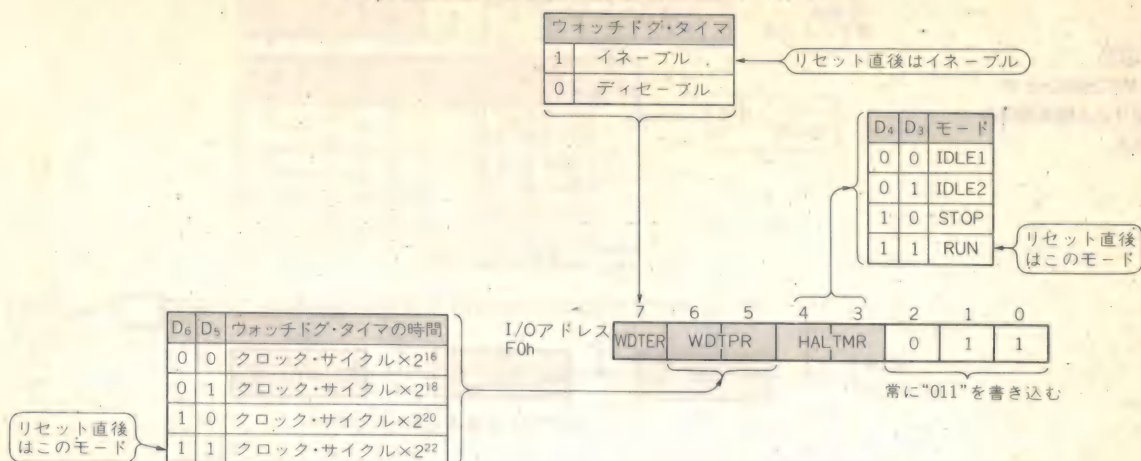
ウォッチドグ・タイマ設定レジスタには、ウォッチドグ機能を使うか使わないかのイネーブル設定ビットと、カウンタをクリアしてからリセットがかかるまでの時間を選択、設定します。

リセットまでの時間は、システム・クロック  $\times 2^{16}$ ,  $2^{18}$ ,  $2^{20}$ ,  $2^{22}$  の 4 種類が選択できます。システム・クロック  $\times 2^{16}$  を設定した場合、クロック 10 MHz では 6.5 ms の周期となります。1 命令の実行クロックを平均 10 クロックとすると、約 6500 ステップ分のプログラムの実行時間に相当します。

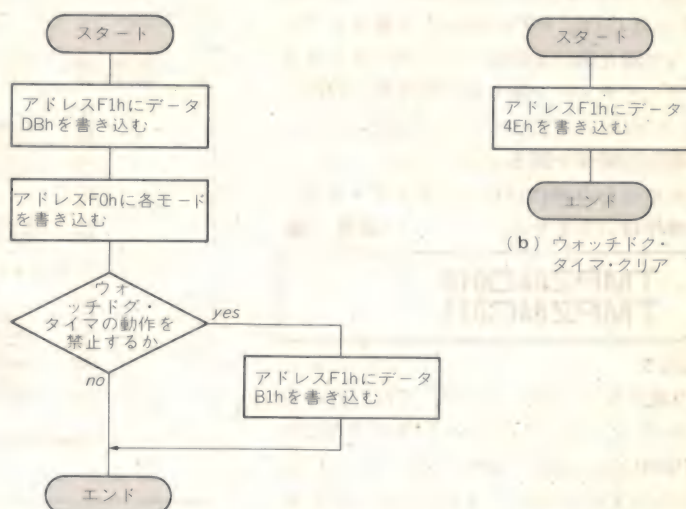
またスタンバイ・モードの設定で説明したように、



〈図3〉スタンバイ・モードとウォッチドグ・タイマの設定



〈図4〉スタンバイ・モードとウォッチドグ・タイマの設定の流れ



(a) スタンバイ・モード/ウォッチドグ・タイマの設定

ウォッチドグ・タイマのレジスタは二重コントロールになっているので同様の手順でレジスタを設定します。

スタンバイ・モードとウォッチドグ・タイマの設定を図3に、設定や解除の流れを図4に示します。またリスト1に設定プログラム例を示します。

### ● ウォッチドグ・タイマの活用

ウォッチドグ・タイマとはマイコンの信頼性を上げるためのものです。ではウォッチドグ・タイマをクリアするためのプログラムは、どのように埋め込んだらよいのでしょうか。

決められた時間以内にクリアするわけですから、タイマ割り込みなどで処理すると簡単そうですが、ウォッチドグ・タイマをタイマ割り込みでクリアするのはナンセンスです。

ウォッチドグ・タイマとは、メイン・ルーチンが、正しく繰り返し実行されているかをチェックするものと

考えてください。よってメイン・ルーチンのループの中の1箇所だけに入れるべきです。しかもサブルーチンなどにせずに直接 OUT 命令を実行したほうがよいでしょう。

また、例えばシステム・クロックが10 MHzのとき、ウォッチドグ・タイマの時間は最長でもシステム・クロック×2<sup>22</sup>時間、約0.4秒となります。つまりメイン・ルーチンを1回ループするときの処理時間は、この時間以内に収まるようにしなければなりません。

例えば工作機械のアームなどは、動き出してから0.5秒もあれば作業員を跳ね飛ばすこともできるでしょう。つまり0.5秒もの長時間(マイコンにとっては長時間と考えるべき)マイコンが暴走していたらたいへんなことになる可能性もあるのです。

### ● 割り込み優先順位

第6章で解説したように、Z80はディジィ・チェー

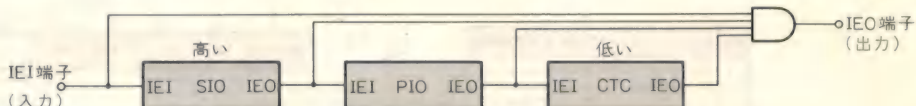
ビット	7	6	5	4	3	2	1	0
F4h番地書き込みのみ	×	×	×	×	×	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

〈図5〉

TMPZ84C015の  
割り込み優先順位の  
設定

優先順位 高 ← 低	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
CTC-SIO-PIO	0	0	0
SIO-CTC-PIO	0	0	1
CTC-PIO-SIO	0	1	0
PIO-SIO-CTC	0	1	1
PIO-CTC-SIO	1	0	0
SIO-PIO-CTC	1	0	1

(a) 割り込み優先順位の設定



(b) "101"を書き込んだときの例

ン方式により割り込み優先順位を制御しています。  
TMPZ84C015にはCTC、PIO、SIOが内蔵されているので、これらの優先順位を設定する必要があります。

これはデジジ・チェーン割り込み優先順位設定レジスタ(アドレスF4h)の下位3ビットで設定します。設定値と優先順位の関係を図5に示します。

この設定のためのI/O操作はウォッチドグ・タイマのような二重操作はいりません。

〈高見 豊〉

## TMPZ84C013 TMPZ84C011

### ● TMPZ84C013

Z80PIOが内蔵されていないだけで、それ以外のI/Oやウォッチドグ・タイマ、スタンバイ・モードなどの設定も、TMPZ84C015と同一です。I/Oアドレスも、TMPZ84C015からPIOのアドレスがなくなっただけで、他のI/Oアドレスに変更はありません。ピン配置を図7と表5に示します。

### ● TMPZ84C011

これはTMPZ84C015からSIOとWDTを取り、さらに8ビット・パラレル入出力ポートを5チャンネル装備した、パラレル入出力を強化したCPUです。図7と表5にピン配置を表6に内蔵I/Oアドレスを示します。またCTCの出力であるZC/TO<sub>3</sub>もありません。

このパラレル・ポートはZ80PIOとは異なり、

### 〈リスト1〉ウォッチドグ・タイマ設定プログラム例

```
HALTMR EQU 0F0H ;ホールド・モード・レジスタI/O
HALTMR EQU 0F1H ;コマンド・レジスタI/O
```

```
D1 LD A,0DBH ;ホールド・モード解除データ
OUT (HALTMR),A ;書き込み
LD A,11111011B ;WDTイネーブル RUNモード
OUT (HALTMR),A
```

```
LD A,04EH ;WDTクリア・データ
OUT (HALTMR),A ;WDTクリア
```

;(a) WDTイネーブル・プログラム

```
D1 LD A,0DBH ;ホールド・モード解除データ
OUT (HALTMR),A ;書き込み
LD A,01111011B ;WDTディセーブル RUNモード
OUT (HALTMR),A
LD A,0B1H ;WDTディセーブル・データ
OUT (HALTMR),A ;書き込み
EI
```

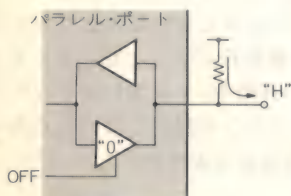
;(b) WDTディセーブル・プログラム

Z80PIOの割り込みを使わないモード3のビット・モード専用のパラレル・ポートと同じような使い方をします。

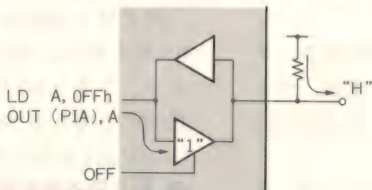
### ● パラレル・ポートの使い方

パラレル・ポートの入出力は1ビット単位で設定でき、対応するチャンネルの入出力設定レジスタの対応するビットを設定します。"0"で入力、"1"で出力です。

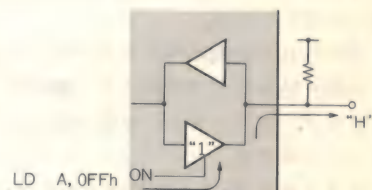
〈図6〉パラレル・ポートの動作



(a) リセット直後



(b) 出力レジスタを"1"にする



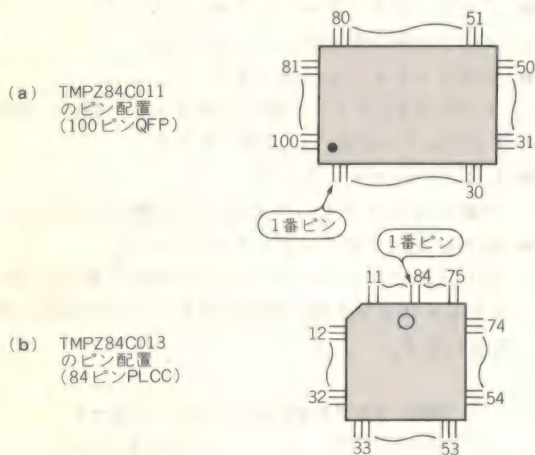
(c) 出力方向に切り替え



〈表5〉 TMPZ84C013/011 のピン配置

ピン名称	TMPZ84C011	TMPZ84C013	ピン名称	TMPZ84C011	TMPZ84C013	ピン名称	TMPZ84C011	TMPZ84C013
D <sub>0</sub>	6	83	CLK/TRG <sub>0</sub>	5	75	CLKIN	—	67
⋮	⋮	⋮	⋮	⋮	⋮	RESET	24	28
D <sub>7</sub>	13	76	CLK/TRG <sub>3</sub>	2	72	INT	14	25
A <sub>0</sub>	29	1	ZC/TO <sub>0</sub>	99	68	NMI	45	56
⋮	⋮	⋮	⋮	⋮	⋮	W/RDY <sub>A</sub>	—	32
A <sub>15</sub>	44	16	ZC/TO <sub>2</sub>	97	70 (ZC/TO <sub>3</sub> : 71)	W/RDY <sub>B</sub>	—	54
PA <sub>0</sub>	68	—	IEI	1	60	SYNC <sub>A</sub>	—	33
⋮	⋮	—	IEO	87	59	SYNC <sub>B</sub>	—	53
PA <sub>7</sub>	61	—	XTAL <sub>1</sub>	49	63	RxD <sub>A</sub>	—	34
PB <sub>0</sub>	76	—	XTAL <sub>2</sub>	48	64	RxD <sub>B</sub>	—	52
⋮	⋮	—	MREQ	17	23	RxC <sub>A</sub>	—	35
PB <sub>7</sub>	79	—	IORQ	18	21	RxC <sub>B</sub>	—	51
PC <sub>0</sub>	60	—	WAIT	22	19	TxC <sub>A</sub>	—	36
⋮	⋮	—	BUSREQ	23	18	TxC <sub>B</sub>	—	50
PC <sub>7</sub>	53	—	BUSACK	21	29	TxD <sub>A</sub>	—	37
PD <sub>0</sub>	96	—	HALT	16	31	TxD <sub>B</sub>	—	49
⋮	⋮	—	RFSH	26	26	DTR <sub>A</sub>	—	38
PD <sub>5</sub>	91	—	EV	15	58	DTR <sub>B</sub>	—	48
PD <sub>8</sub>	89	—	TEST	86	—	RTS <sub>A</sub>	—	34
PD <sub>7</sub>	88	—	ICT	—	42, 44	RTS <sub>B</sub>	—	47
PE <sub>0</sub>	85	—	MS <sub>1</sub>	47	—	CTS <sub>A</sub>	—	40
⋮	⋮	—	MS <sub>2</sub>	46	—	CTS <sub>B</sub>	—	46
PE <sub>7</sub>	78	—	CLK	52	66 (CLKOUT)	DCD <sub>A</sub>	—	41
M <sub>I</sub>	25	17				DCD <sub>B</sub>	—	45
R <sub>D</sub>	19	30				V <sub>CC</sub>	27, 51, 90	43, 84
W <sub>R</sub>	20	20				V <sub>SS</sub>	28, 50, 77, 100	22, 62

〈図7〉 TMPZ84C011 と 013 のピン配置



〈表6〉 TMPZ84C011 内蔵 I/O のポート・アドレス

内蔵 I/O デバイス	チャンネル	I/O アドレス
CTC	チャンネル 0	10h
	チャンネル 1	11h
	チャンネル 2	12h
	チャンネル 3	13h
パラレル I/O	チャンネル A 入出力設定レジスタ	54h
	チャンネル B 入出力設定レジスタ	55h
	チャンネル C 入出力設定レジスタ	56h
	チャンネル D 入出力設定レジスタ	34h
	チャンネル E 入出力設定レジスタ	44h
	チャンネル A データ	50h
	チャンネル B データ	51h
	チャンネル C データ	52h
	チャンネル D データ	30h
	チャンネル E データ	40h

リセット時はすべてのポートの設定が入力になっています。またこのパラレル・ポートは、ポートが入力に設定されていても、書き込み動作をすると出力レジスタにデータがラッチされます。それからポートを出力に設定すると、書き込んだ値が即座に出力されるようになっています。

よって図6のように端子をプルアップしておくと、リセット直後はポートが入力に設定されるので、端子

の状態としては H レベルを出力することになります。

まず CPU がリセットされると PIO の出力レジスタはクリアされ “0” になります。このあと “1” を書き込むと出力レジスタにラッチされ、次に方向を出力に設定しても端子の状態は H レベルを保ちます。これにより、Z80PIO や 8255 などと問題になる、初期化時に瞬間的に L レベルが出力されるという問題がありません。

# Z84C15/Z84C11

Z84Cxx シリーズ(ザイログ)は、TMPZ84C015(東芝)に対応する Z84C15 と、TMPZ84C011 に対応する Z84C11 の 2 種類がよく使われます。内蔵 I/O の種類、アドレス、設定方法などは、Z84C15 は TMPZ84C015 と、Z84C11 は TMPZ84C011 とまったく同じです。

しかしこの 2 種類には、次の機能が追加されています。

- ▶ パワー ON リセット
- ▶ プログラマブル・ウェイト・ステート・ジェネレータ
- ▶ プログラマブル・アドレス・デコーダ
- ▶ 32 ビット CRC ジェネレータ・チェッカ (SIO チャネル A のみ)
- ▶ SIO 送受信クロック・シュミット・トリガ入力
- ▶ CGC 出力周波数 1/1, 1/2 の選択が可能
- ▶ EV モード時、外部回路不要

また当然のことながら、Z84C11 には SIO が内蔵されていないので、32 ビット CRC や送受信クロック入力のシュミット・トリガ機能はありません。またプログラマブル・アドレス・デコーダの機能也没有ありません。

PIO, CTC, SIO, WDT のアドレスは、TMPZ84C015 や 011 とまったく同じですが、これ以外に、上記の機能をコントロールするための、システム・コントロール・レジスタが追加されています(表 7)。

ここではとくに、パワー ON リセット機能と、プログラマブル・ウェイト・ステート・ジェネレータ、プログラマブル・アドレス・デコーダについて説明します。

## ● パワー ON リセット

Z84C15/C11 のリセット端子はオープン・ドレイン

〈表 7〉 Z84C15/C11 に追加されたシステム・コントロール・レジスタの I/O アドレス

SCRП(システム・コントロール・レジスタ・ポインタ)	EEh
SCDP(システム・コントロール・データ・ポート)	EFh

〈表 8〉 コントロール・レジスタのアドレス

00000000b (00h)	WCR(ウェイト・コントロール・レジスタ)
00000001b (01h)	MWBR(メモリ・ウェイト・バウンダリ・レジスタ)
00000010b (02h)	CSBR(チップ・セレクト・バウンダリ・レジスタ)
00000011b (03h)	MCR(Misc コントロール・レジスタ)

端子で、切り離し可能なリセット IC が内蔵された端子と考えることができます。電源投入時は、内蔵のリセット IC がイネーブルに設定されるので、電源電圧が約 2.2 V を越えてから約 25~75 ms の期間、L レベルを出力します。また立ち下がりエッジを検出すると、システム・クロックで 16 クロックの時間、L レベルを出力します。

またシステム・コントロール・レジスタを設定することで、リセット IC をディセーブルに設定することも可能です。この場合は通常の Z80CPU と同様、システム・クロックで 3 クロック以上の時間、リセット端子に L レベルを入力することで、リセットをかけることができます。

## ● ウェイト・ステート・ジェネレータ

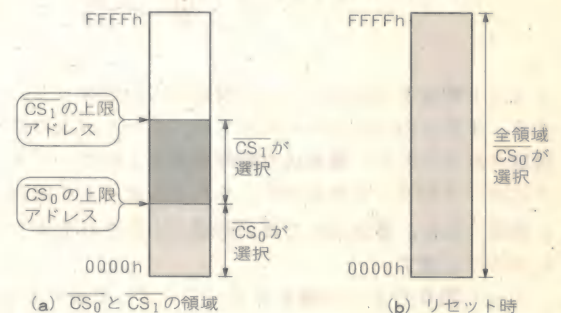
Z84C15/C11 は、各サイクルに外付け回路なしでウェイトを追加することができます。

このとき、WAIT 端子によるウェイト・コントロールは、ウェイト・ステート・ジェネレータによるウェイト・ステート終了後にサンプリングされます(割り込みアクノリッジ時を除く)。

以下のサイクルのときにウェイトを入れることができます。

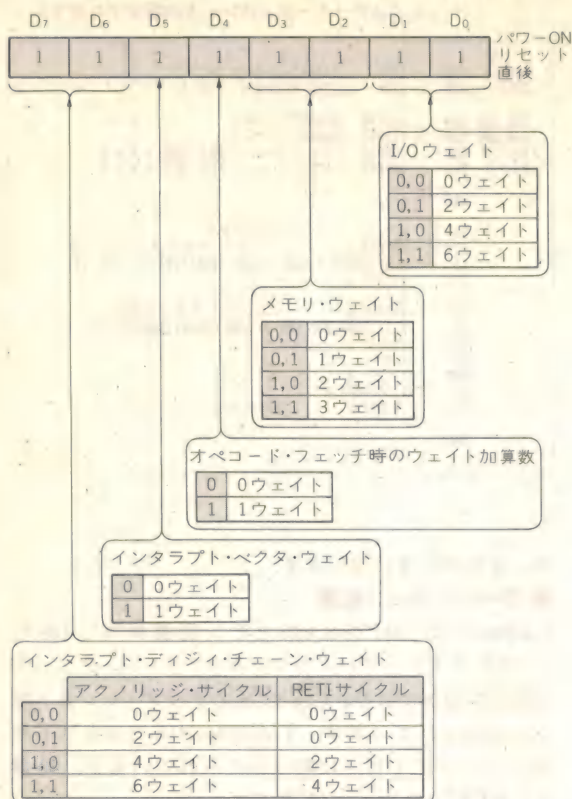
- ▶ メモリ・アクセス・サイクル  
メモリの読み書き時のウェイトです。
- ▶ オペコード・フェッチ・サイクル  
オペコード・フェッチ時の合計ウェイト数は、メモリ・ウェイトにさらに加算されます。
- ▶ I/O アクセス・サイクル  
内蔵 I/O をアクセスするときには挿入されません。
- ▶ 割り込みベクタ・フェッチ・サイクル  
割り込みアクノリッジ・サイクル中の、割り込みベクトル・フェッチ時、 $\overline{\text{IORQ}}$  が L レベルの期間に挿入されます。

〈図 8〉 プログラマブル・アドレス・デコーダ

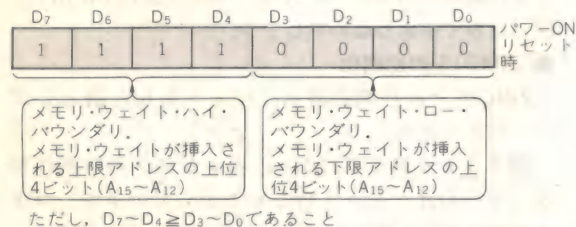




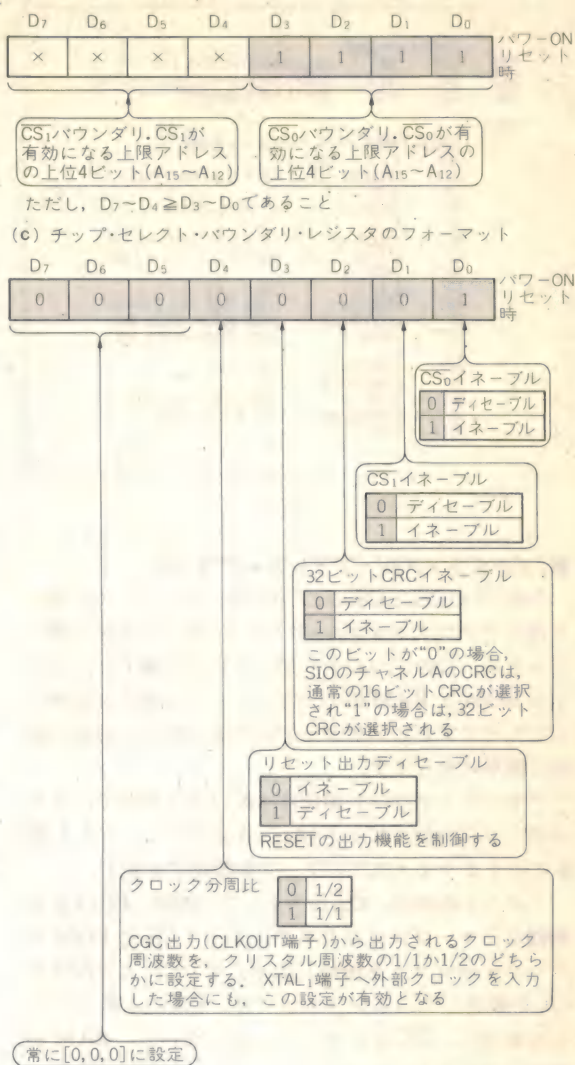
〈図9〉 各種システム・コントロール・レジスタの設定



(a) ウェイト・コントロール・レジスタのフォーマット



(b) メモリ・ウェイト・バウンダリ・レジスタのフォーマット



(d) Miscコントロール・レジスタのフォーマット

## ▶ 割り込みアクノリッジ・サイクル

割り込みアクノリッジ・サイクルにおいて、 $\overline{MI}$ がLレベルになってから、 $\overline{IORQ}$ がLレベルを出力するまでの間に挿入されます。さらに、RETI(割り込み復帰)サイクル中にも挿入されます。これらは、デジィ・チェーン接続された周辺LSIへのアクセス時間を確保するために挿入されます。

リセット後はすべてのサイクルに、最大数のウェイトが自動的に挿入されます。そして16回目のオペコード・フェッチ・サイクルから、ウェイト・ステートの自動挿入はなくなります。

よって、アクセス速度の遅いメモリなどを使用する場合は、リセット後16ステップ以内に、ウェイト・ステート・ジェネレータに対して各サイクルのウェイト

数を設定しなければなりません。

また、メモリ・アクセス・サイクルに対するウェイトは、ウェイトを入れるアドレス範囲を指定できます。つまり、RAMには高速なメモリを使っているが、ROMが遅いという場合、ROM領域だけにウェイトを入れることができます。

アクセス速度とメモリ・アクセス時のウェイト数の目安としては、システム・クロックが6 MHzのCPUの場合、200 ns以上のメモリを使用するなら1ウェイト、それより速いメモリならウェイトの必要はありません。システム・クロック10 MHzのCPUの場合、200 ns以上のメモリなら2ウェイト、200 ns~100 nsなら1ウェイト、100 ns以下ならウェイトなしとなります。

## 「リスト2」 システム・コントロール・レジスタの設定例

```

; I/Oアドレス設定
SCRIP EQU 0EEH ;システム・コントロール・レジスタ・バンクA
SCDP EQU 0EFH ;システム・コントロール・データ・ポート
IPR EQU 0F4H ;割り込み優先順位設定

START:
; Z84C15 内部コントロールレジスタ初期化
XOR A
OUT (SCRIP),A ;ポイント0 WAIT Ctrl
LD A,0
OUT (SCDP),A ;ノン・ウェイト
LD A,1
OUT (SCRIP),A ;ポイント1 WAIT/バウンダリ
LD A,0F0H ;全アドレス ウェイト無し
OUT (SCDP),A
LD A,2
OUT (SCRIP),A ;ポイント2
LD A,11110111B ;/CS0 /CS1 コントロール
OUT (SCDP),A ;/CS0 0000h~7FFFh ROM
LD A,3
OUT (SCRIP),A ;ポイント3
LD A,00000011B ;/CS0 /CS1 イネーブル
OUT (SCDP),A ;PIO > CTC > SIO
LD A,00000100B
OUT (IPR),A

```

## 「リスト3」 プログラマブル・アドレス・デコーダとウェイト・ステート・ジェネレータの設定プログラム

```

SCRIP EQU 0EEH ;システム・コントロール・レジスタ・バンクA
SCDP EQU 0EFH ;システム・コントロール・データ・ポート

ROM_END EQU 03FFFF ;ROM終了アドレス
RAM_END EQU 07FFFF ;RAM終了アドレス
WAIT_S EQU 00000H ;メモリ・ウェイト挿入 開始アドレス
WAIT_B EQU 0FFFFH ;メモリ・ウェイト挿入 終了アドレス

ORG 0

LD A,1 ;ポイント1
OUT (SCRIP),A ;ウェイト・バウンダリ・レジスタ
LD A,0+((WAIT_B SHR 8) AND 0F0H)+(WAIT_S SHR 12)
OUT (SCDP),A
LD A,2 ;ポイント2
OUT (SCRIP),A ;/CS0 /CS1 アドレス設定
LD A,0+((RAM_END SHR 8) AND 0F0H)+(ROM_END SHR 12)
OUT (SCDP),A
LD A,3 ;ポイント3
OUT (SCRIP),A
LD A,00000011B ;/CS0 /CS1 イネーブル
OUT (SCDP),A

END

```

### ● プログラマブル・アドレス・デコーダ

Z84C15 は $\overline{CS_0}$ と $\overline{CS_1}$ の二つのチップ・セレクト端子を備えたプログラマブル・アドレス・デコーダを内蔵しています。この二つのチップ・セレクト端子に、それぞれROMとRAMのチップ・セレクト端子を接続することでアドレス・デコーダが不要になり、回路が非常に簡単になります。

アドレス・デコード単位は4Kバイト単位で、アドレス・バス上位4ビット( $A_{15} \sim A_{12}$ )をデコードし、図8に示すようなメモリ・マップを設定できます。

リセット直後は、 $\overline{CS_0}$ がアドレス0000h~FFFFhの範囲でイネーブルになります。よって $\overline{CS_0}$ にROMのチップ・セレクトを接続します。また $\overline{CS_1}$ にRAMの $\overline{CS}$ を接続しているシステムでは、 $\overline{CS_0}$ の上限アドレスを設定し、 $\overline{CS_1}$ をイネーブルにしないと、RAMが選択されません。

### ● システム・コントロール・レジスタの使い方

プログラマブル・ウェイト・ステート・ジェネレータやアドレス・デコーダなどの設定は、システム・コントロール・レジスタをアクセスすることで設定します。

システム・コントロール・レジスタは、SCRIP(システム・コントロール・レジスタ・ポイントA: I/OアドレスEEh)にアクセスしたいコントロール・レジスタのアドレスを指定し、SCDP(システム・コントロール・データ・ポート: I/OアドレスEFh)で読み書きするという手順を取ります。

各種コントロール・レジスタのアドレスとコントロール・データの意味を表8と図9に、システム・コントロール・レジスタの設定方法の例をリスト2に示します。またプログラマブル・アドレス・デコーダとメモリ・アクセスにウェイトを入れる領域の設定を、アセンブラの算術演算子を使って自動的に計算するプログ

ラムをリスト3に示します。

### ● Z84C15 のピン配置

Z84C15 はTMPZ84C015とピン配置が同じ100ピンQFPですが、プログラマブル・アドレス・デコーダの機能の追加により、 $\overline{CS_0}$ と $\overline{CS_1}$ のチップ・セレクト端子が追加されています。TMPZ84C015ではテスト用端子だったICTピンに割り当てられています。40番ピンが $\overline{CS_0}$ 、42番ピンが $\overline{CS_1}$ です。

また電気的特性についても、ほぼTMPZ84C015と同じとみてかまわないでしょう。

### ● Z84C15 の初期化

Z84C15では初期化時に、ちょっとした落とし穴(?)があります。

図9をよく見てください。各レジスタの初期状態をパワーONリセット時として示しています。つまり電源を投入したときにしかクリアされないのです。

ということは、最初に電源を投入してこれらのレジスタを一度だけ設定すれば、次にリセット・スイッチによりリセットしても、その設定がクリアされずに残るわけです。

これでよく失敗するのは、プログラムの開発中などのときです。一度初期化がうまくいくと、その設定がリセット後も残りますから、その後のプログラムの修正で、実は初期化部分を誤って変えてしまっても、リセット・スイッチによるリセットではそのまま動作してしまうわけです。

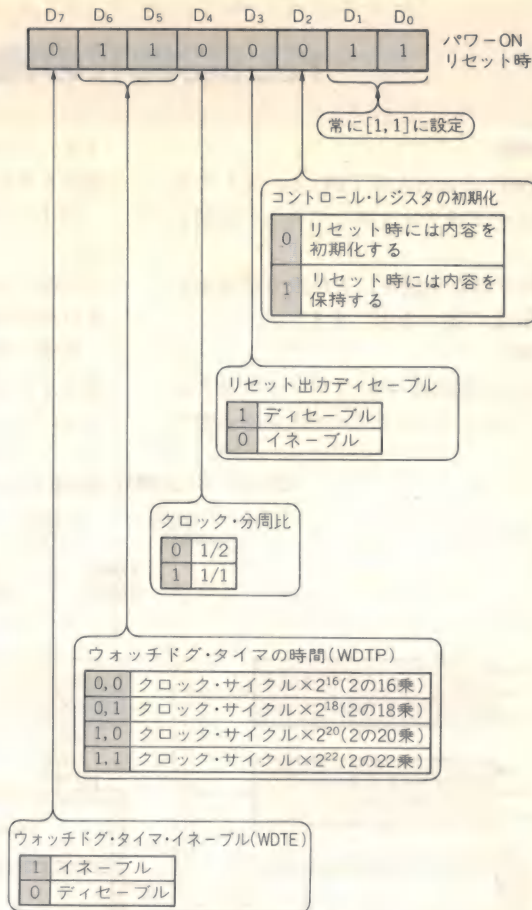
こうして完成したと思ってROM化し、改めて電源ONで走らせると…ということになりかねません。注意しましょう。

### ● 1/2分周と1/1分周クロック

これは例えば、5MHz版のCPUに10MHzの水素振動子を接続し、普段は1/2分周の5MHzで動作



〈図 10〉  
Z84C11 のウォッチドグ・  
タイマ・レジスタ  
(I/O アドレス：F0h)



させ、ターボ・モードで1/1分周の10MHzにできる  
…というものではありません。

1/1分周を使うときは、クロック周波数と同じ周波数の水晶振動子を使い、少しでも低消費電力にしたいときに半分のクロック周波数で動作させ、処理速度を優先させたいときに1/1分周に設定するという目的で使います。

よって1/1分周に設定したときのクロック周波数が、CPUの最高動作クロック周波数を超えないように周波数を選択します。

#### ● Z84C11のピン配置

これもピン配置はTMPZ84C011と同じですが、ウォッチドグ・タイマ機能が追加されているので、その出力端子である $\overline{\text{WDTOUT}}$ 端子が、パラレル・ポートのチャンネルEのビット7(78番ピン)のピンと兼用に割り当てられています。

ウォッチドグ・タイマがイネーブルに設定されると、 $\overline{\text{WDTOUT}}$ になります。このときチャンネルEのビッ

ト7に書き込まれた値は、 $\text{PE}_7/\overline{\text{WDTOUT}}$ ピンには影響は与えませんが、出力ラッチには書き込まれています。また読み出しをしたときは、 $\overline{\text{WDTOUT}}$ の出力状態が読み込まれます。リセット時は、ウォッチドグ・タイマはディセーブルです。

またプログラマブル・アドレス・デコーダなどの機能がないので、システム・コントロール・レジスタのCSBRとMCRはありません。またこのため内蔵のリセットICのイネーブル・コントロールや、CGCクロックの分周比の設定のビットが、モード設定レジスタ(アドレスF0h)に割り当てられているので注意します(図10)。

また図10でわかるように、これもC15と同様に、リセット時にコントロール・レジスタの状態を保持することもできます。しかしプログラマブルに設定できる点の違いです。

クロック周波数の分周設定についてもC15と同様です。

〈菅原尚伸〉

# KL5C8012の概要

## ● KL5C8012 の特徴

ここではKL5C8012とはどんなCPUか、そして気になるその処理スピードをチェックしてみたいと思います。

まず図11に内部ブロック図を、また表9に従来のZ80やその周辺LSIとの違いを示します。

## ● CPU コア KC80

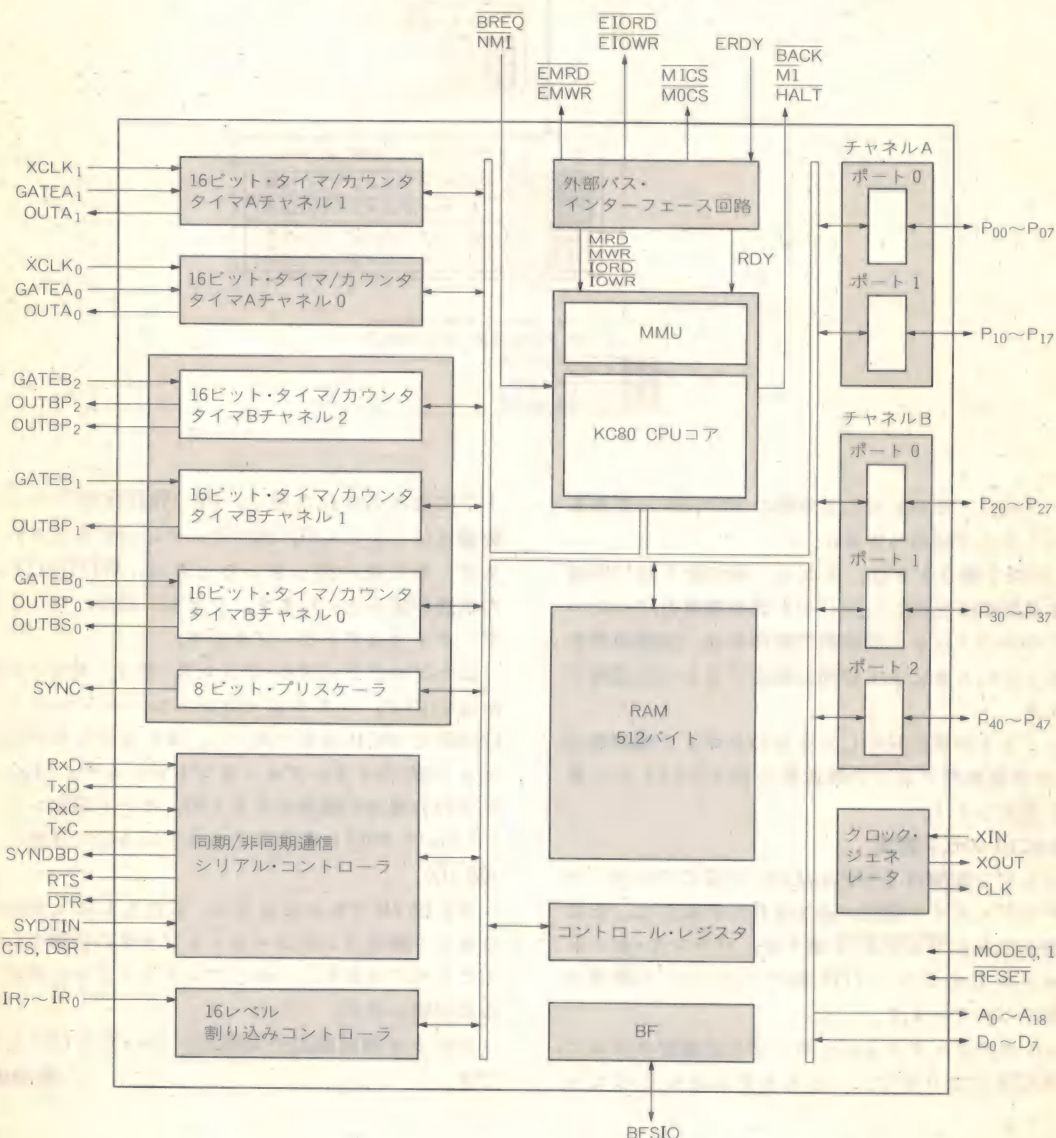
KL5C8012の最大の特徴は、Z80とソフトウェアがバイナリ・レベルでフル・コンパチブルだという点でし

よう、そのため、基本的には従来からのソフトウェア資産をそのまま使うことが可能です。

CPUの最大クロック周波数は10MHzです。しかし、一つ一つの命令実行に必要なクロック数が、Z80と比較して1/2~1/11と少なくなっています(表10)。また最小命令実行時間は1クロックで0.1μsです。

外部メモリや外部I/Oのアクセス・タイミングも高速化されています。ノン・ウェイト時は、1クロックでリード/ライト・サイクルを実行することになります

(図11)<sup>(4)</sup> KL5C8012の内部ブロック





(図 12).

Z80 は最少命令実行時間が4クロックで、システム・クロック 10 MHz 時は 0.4  $\mu$ s です。

したがって、同一動作クロック数の Z80 と比較しても単純計算で 4 倍高速となっています。

CPU の信号として外部に出力されているものは、M1, HALT, NMI, ERDY, BREQ (BUSREQ), BACK (BUSACK) です (括弧内は Z80 での名称)。

また Z80 の RD, WR, MREQ, IORQ は出力されていませんが、そのかわり、メモリ・アクセス用に

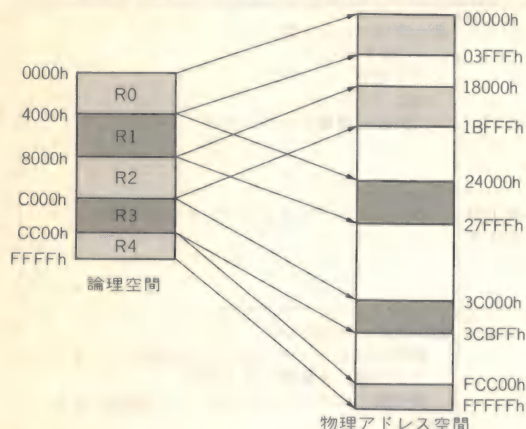
〈表 9〉 KL5C8012 と Z80 の比較

	KL5C8012	Z80 と周辺 LSI
CPU	命令実行クロック数 1~7 クロック メモリ空間 一度に扱える範囲は 64K バイト MMU を内蔵し、バンク切り替えによって 最大 512K バイトまでアクセス可能 チップ内部に 512 バイトの RAM を内蔵 ウェイト 外部メモリ・アクセスに 1 クロック、外部 I/O アクセスに 2 クロック、ウェイト挿入をプログラムで制御可能 ハードウェアによってさらに延長可能	命令実行クロック数 4~23 クロック メモリ空間 64K バイト MMU は準備されていない ウェイト ハードウェアによってウェイト挿入可能
割り込みコントローラ	16 入力 (内部 8, 外部 8) モード 2 対応 エッジ、レベル入力切り替え可 多重割り込み可能 優先順位変更可能	Z80 ファミリにはない 周辺 LSI にモード 2 割り込み制御のハードウェアを内蔵、多重割り込み可能 優先順位はディジィ・チェーンの接続による 8259A モード 0 使用 8 入力 エッジ、レベル入力切り替え可 多重割り込み可能 優先順位変更可能
タイマ/カウンタ	タイマ/カウンタ A 16 ビット・カウンタ 2 チャネル カスケード接続により 32 ビット化可能 4 種類の動作モードをもつ タイマ/カウンタ B 8 ビット・プリスケアラ付き 16 ビット・カウンタ 3 チャネル チャネル 1 はボーレート設定用に使うためシリアル・ポートの送受信クロックに内部で接続可能 4 種類の動作モードをもつ	Z80CTC 8 ビット・プリスケアラ付き 8 ビット・カウンタ 4 チャネル 2 種類の動作モードをもつ 8253/8254 16 ビット・カウンタ 3 チャネル 6 種類の動作モードをもつ
シリアル・ポート	8251A と同一機能 1 チャネル 非同同期式・同期式通信可能	Z80SIO 2 チャネル内蔵 非同同期式・同期式・SDLC 通信可能 CRC 生成・チェック機能内蔵 8251A 1 チャネル 非同同期式・同期式通信可能
パラレル・ポート	パラレル・ポート A 8 ビット 2 チャネル 1 ビット単位で入出力の方向制御が可能 出力切り替え時のデータ・プリセット機能 パラレル・ポート B 8 ビット 3 チャネル 4 ビット単位で入出力の方向制御が可能 ビット単位のセット/リセットが可能 出力切り替え時のデータ・プリセット機能	Z80PIO 8 ビット 2 チャネル ハンドシェイクによる入出力機能内蔵 ビット・モード時 (単純な入出力ポート), I ビット単位で入出力の方向制御が可能 ビット・モード時, 入出力ポートのデータ値により割り込み発生可能 8255A 8 ビット 3 チャネル ストロープ入出力機能内蔵 単純入出力ポート時, ビット単位のセット/リセットが可能

〈表 10〉 実行クロック数の比較

命令の例	KL5C8012	Z80
LD B, C	1 クロック	4 クロック
ADD HL, BC	1 クロック	11 クロック
DEC DE	1 クロック	6 クロック
POP AF	3 クロック	10 クロック
JR Z, +20h	3 クロック	12 クロック

〈図 13〉<sup>(4)</sup> MMU の論理アドレスと物理アドレス



$\overline{\text{EMRD}}$ ,  $\overline{\text{EMWR}}$ , I/O アクセス用に  $\overline{\text{EIORD}}$ ,  $\overline{\text{EIOWR}}$  が用意されています。

#### ● MMU を内蔵

MMU(メモリ・マネージメント・ユニット)を内蔵することにより、アドレス空間は 1 M バイト(物理アドレス)に拡張されています。ただしアドレス・バスが 19 本しかないので、外部メモリは最大 512 K バイトまで接続できます(マキシマム・モード時)。

また一度に扱えるメモリ容量は Z80 と同じ 64 K バイト(論理アドレス)です。よって 64 K バイトをこえるデータを扱うために、バンク切り替えを行う必要があります。

論理アドレス空間は図 13 に示すように五つの領域(R0~R4)に分割されています。

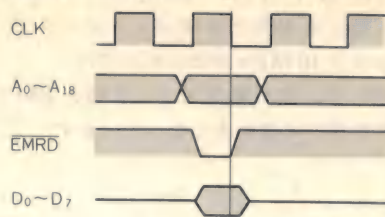
#### ● メモリ・マップ

外部メモリの接続方式はノーマル・モードとマキシマム・モードの 2 種類あり、MODE 端子で設定します。

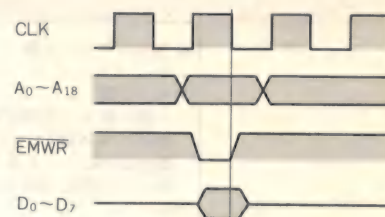
ノーマル・モードのときは、アドレス・バス  $A_{18}$ ,  $A_{17}$  はチップ・セレクト端子として動作します。したがって  $A_0 \sim A_{16}$  までの 128 K バイトのメモリを二つ、合計 256 K バイトまでを制御できます。

マキシマム・モードのときは、本来のアドレス・バスとして動作するので、 $A_0 \sim A_{18}$  までの 512 K バイトとなります。

〈図 12〉<sup>(4)</sup> KL5C8012 のメモリ・アドレス



(a) 外部メモリ・リード・サイクル



(b) 外部メモリ・ライト・サイクル

また、内蔵 RAM は物理アドレスで FFE00h ~ FFFFFh に配置されています。この RAM は後述のウェイトの設定にかかわらず、0 ウェイトでアクセスされます。

#### ● ウェイト機能

メモリや I/O の中にはスピードが遅いものもあります。そのためのウェイト制御回路を内蔵しており、外部メモリ・アクセスに 1 ウェイト、外部 I/O アクセスに 2 ウェイト入れることができます。

#### ● デバッグ機能

KL5C8012 の特徴として、デバッグ用の専用端子と回路を内蔵している点があげられます。

この専用端子にバグ・ファインディング・アダプタ BFA8001(川崎製鉄)を接続し、パソコンの RS-232-C ポートと接続することによって、ブレーク・ポイントの設定や内部レジスタの読み書き、メモリの読み書きが行えます。

ユーザのメモリや I/O 空間とは異なる空間に置かれるため、ユーザ側の制限はまったくありません。またハードウェア・ブレーク対応など、ROM 上のプログラムにもブレークをかけることができ、ICE 感覚で使用できます。

#### ● スピードの比較

ここでは KL5C8012 の評価ボード SMC01 を使い、Z80 と同一プログラムを実行させたときの処理時間の違いで実行スピードを比較してみます。

比較のための Z80 ボードには TK-Z80(東洋リンクス)を使います。クロック周波数は 4.9152 MHz です。SMC01 の動作クロック数は 10 MHz なので、テスト・プログラムの実行時間を倍にすれば、クロック 5



〈表 11〉 実行速度の違い

プログラム 番号	SMC01 (10MHz)単位 ms	TK-Z80 (4.9152MHz)単位 ms	同一クロックでの スピード比(TK-Z80を1)
①	39	320	4.03 倍
②	0.1972	1.59	3.96 倍
③	0.162	1.14	3.46 倍
④	0.5943	4.16	3.44 倍
⑤	2.57	17.74	3.39 倍
⑥	10.7	74.1	3.40 倍
⑦	21.9	150.7	3.38 倍

MHz 相当の実行時間になります。

#### ● テスト・プログラムの結果

テストに使用したプログラムは以下のとおりです。

- ① 単純ループ(65536 回)
- ② 16 ビット、8 データのソート
- ③ 8 ビット、256 データからのサーチ
- ④ 32 ビット×32 ビット
- ⑤ 64 ビット÷64 ビット
- ⑥ loge(X)
- ⑦ デシベル値の計算

表 11 に実測した実行時間と、そのスピード比を示します。

この結果からすると、同一クロック数では3〜4 倍のスピードアップになっていることがわかります。

①がいちばん速くなりましたが、これは変数をレジスタだけでプログラムしており、ほかはメモリ・アクセスが入っているからだと考えられます。

例えば、LD A, B は Z80 とくらべて 1/4 になるのに対し、LD A, (8000H) は 3/10 となります。

今回のテストでは I/O 入出力を含まないプログラムでしたが、実際にコントローラとして使うには I/O の処理が入ります。

しかし、たとえ外付け I/O を増設し、ウェイトを入れたとしても、I/O 入出力を割り込みで起動させれば、I/O で待たされる間にほかの処理を行うことができるので、全体の処理能力を高めることができるでしょう。

#### ● 使用上の注意点

しかし注意点もあります。まず 100 ピン QFP パッケージにこれだけの機能ピンを装備することは当然無理で、兼用端子が多いため同時に使えない機能があるということです。

また Z80 ベースの CPU とはいえ、バス・サイクルやデジィ・チェーン接続用端子などの関係で、Z80 ファミリー LSI を直接外付けすることは難しそうです。

〈菊地恭徳〉

## KL5C8012の使い方

すでに説明したように、KL5C8012 は多機能な周辺 LSI を内蔵していますが、100 ピン・パッケージに収める関係で、マルチプレクス(兼用ピン化)されているピンがあり、同時に使えない場合があります。表 12 に内蔵 I/O のアドレスを、図 14 に兼用ピンになっている入出力ピンの選択レジスタを示します。

また、ここでは KL5C8012 を使った市販マイコン・ボード TC8012(田中電子)と UKC82/01(ユニテック電子)を例にあげて説明します。

#### ● リセット直後の状態

まずリセット直後の状態を知っておかなくてはなりません。注意すべき点は次の 4 点です。

- ・ MMU は一部を除いて外部 ROM を論理アドレス空間に割り付けてしまう
- ・ マルチプレクスされた兼用端子は、すべてパラレル・ポートに選択されている
- ・ ウェイトは外部メモリは 1 ウェイト、外部 I/O は 2 ウェイト
- ・ MODE 端子ピンの状態でノーマル・モード、マキシマム・モード、バグファインディング・モードを決定

したがってリセット後に必要に応じて、内蔵の制御レジスタを設定しなします。

#### ● MMU の初期化

まず外部 RAM を使用する場合は、MMU の設定をしなくてはなりません。

例にあげた二つのボードでは、MODE 端子がノーマル・モードに設定されているので、ここでは MMU 設定など、ノーマル・モードを例に説明します。

ノーマル・モードでは物理アドレス空間(1M バイト)うちの、00000h 番地から 128K バイト(M0CS でアクセス)と、0E0000h 番地から 128K バイト(M1CS でアクセス)を、論理アドレス空間の 64K バイトに、五つの領域に分けて割り当てることのできるモードです。

リセット直後は外部 RAM を使うために MMU を設定します。とりあえず Z80 の標準的な使い方と同じように、

- ・ ROM 領域 32K バイト: 0000h~7FFFh
- ・ RAM 領域 32K バイト: 8000h~0FFFFh

と割り付けるためのもっとも簡単な設定をリスト 4

〈表 12〉<sup>(4)</sup> KL5C8012 内蔵 I/O ポート

I/O アドレス	内蔵 I/O	チャンネル
00h	KC82 MMU	境界/ベース・レジスタ 1
01h		ベース・レジスタ 1
02h		境界/ベース・レジスタ 2
03h		ベース・レジスタ 2
04h		境界/ベース・レジスタ 3
05h		ベース・レジスタ 3
06h		境界/ベース・レジスタ 4
07h		ベース・レジスタ 4
08h-0Fh	川崎製鉄使用予約	
20h	タイマ/ カウンタ B	チャンネル 0 カウンタ
21h		チャンネル 0 コントロール
22h		チャンネル 1 カウンタ
23h		チャンネル 1 コントロール
24h		チャンネル 2 カウンタ
25h		チャンネル 2 コントロール
26h	川崎製鉄使用予約	
27h	川崎製鉄使用予約	
28h	タイマ/ カウンタ A	チャンネル 0 カウンタ
29h		チャンネル 0 コントロール
2Ah		チャンネル 1 カウンタ
2Bh		チャンネル 1 コントロール
2Ch	パラレル A	ポート 0 データ
2Dh		ポート 0 方向制御レジスタ
2Eh		ポート 1 データ
2Fh		ポート 1 方向制御レジスタ
30h	パラレル B	ポート 0 データ
31h		ポート 1 データ
32h		ポート 2 データ
33h		コントロール
34h	割り込み コントローラ	LERL/PGRl
35h		LERH/PGRH
36h		IMRL
37h		VTAR/IMRH
38h	シリアル・ ポート	データ
39h		コマンド/ステータス
3Ah	システム 制御レジスタ	SCR0
3Bh		SCR1

〈リスト 4〉 ROM32K, RAM32K の MMU の設定例

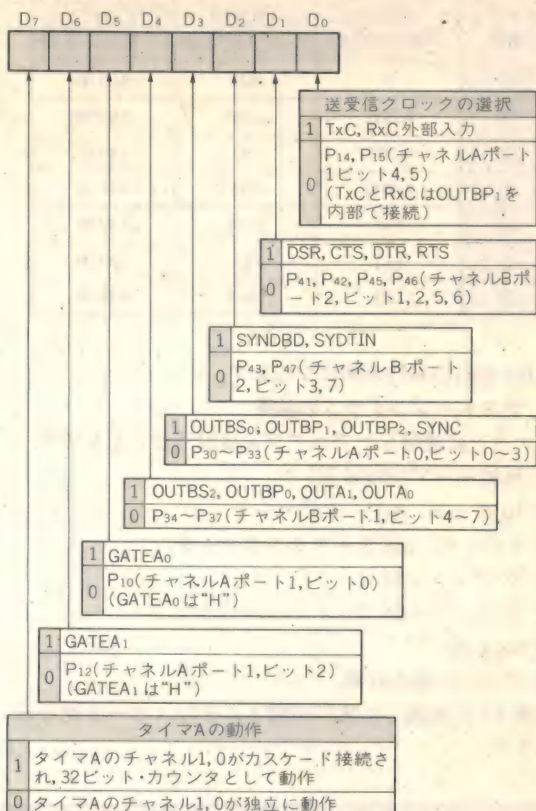
***** MMU設定(ROM32K/RAM32K) *****		
LD	A, 1FH	
OUT	(06H), A	

に示します。MMU レジスタ BBR<sub>4</sub>に 1Fh を書き込むだけです。このときは図 11 でいうところの R0 と R4 しか有効となりません。

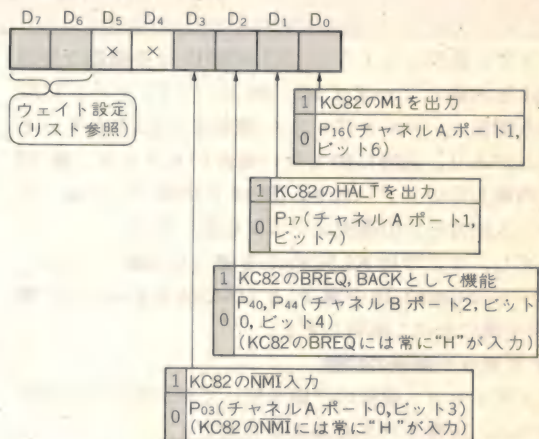
大容量 RAM を使えば、余っている R1~R3 を使ってバンク切り替えを行うこともできます。ラベルに任意の値を設定すれば、アセンブル時に自動的に計算し

〈図 14〉 兼用ピン選択レジスタ

SCR0(I/O アドレス: 3AH)



SCR1(I/O アドレス: 3BH)



て MMU を設定してくれるプログラムをリスト 5 に示すので参考にしてください。

#### ● メモリのアクセス・タイムについて

システム・クロックが 10 MHz の場合、アクセス・タイム 70 ns の ROM や RAM が用意できるなら、リスト 6 の設定でウェイト・コントローラが挿入するウ



## 〈リスト5〉MMU 割り当て計算プログラム

```
*****
; MMU 割り当ての自動計算例
*****
```

;MMUのI/Oアドレス定義

```
BBR1 EQU 00H ;MMU REG BBR1
BR1 EQU 01H ; BR1
BBR2 EQU 02H ; BBR2
BR2 EQU 03H ; BR2
BBR3 EQU 04H ; BBR3
BR3 EQU 05H ; BR3
BBR4 EQU 06H ; BBR4
BR4 EQU 07H ; BR4
```

;物理7D 1先頭値の定義(R1~R4まで割り当てる物理7D 1の先頭を10Hとして定義する)

```
R1_PS EQU 00100H ;各値の下限は00000H,
R2_PS EQU 0F440H ;上限は0FFBCH未満
R3_PS EQU 0F680H ;ステップは40Hとする
R4_PS EQU 0F800H ;0F040H以下の値を定義する事
```

;論理7D 1先頭値の定義(R1~R4まで割り当てる論理7D 1の先頭を定義する)

```
;無効設定をするためには、Bnに3FHを設定するためには、
;10000Hをここで定義する必要があるが、0FFFFHを超えるデータ
;の定義ができないので、便宜的にRn.LS=>Rn+1.LSと定義する
R1_LS EQU 01400H ;ステップは400Hとする
R2_LS EQU 04000H ;
R3_LS EQU 08800H ;
R4_LS EQU 0C000H ;
```

```
_B1 EQU ((R1_LS AND 0FC00H) SHR 10) - 1
_A1 EQU ((R1_PS AND 0FFC0H) - ((R1_LS AND 0FC00H) SHR 4)) SHR 6
_B2 EQU ((R2_LS AND 0FC00H) SHR 10) - 1
_A2 EQU ((R2_PS AND 0FFC0H) - ((R2_LS AND 0FC00H) SHR 4)) SHR 6
_B3 EQU ((R3_LS AND 0FC00H) SHR 10) - 1
_A3 EQU ((R3_PS AND 0FFC0H) - ((R3_LS AND 0FC00H) SHR 4)) SHR 6
_B4 EQU ((R4_LS AND 0FC00H) SHR 10) - 1
_A4 EQU 03C0H ;固定データ
```

```
_BBR1 EQU (((_A1 AND 03H) SHL 6) OR _B1)
_BR1 EQU ((_A1 AND 03FCH) SHR 2)
_BBR2 EQU (((_A2 AND 03H) SHL 6) OR _B2)
_BR2 EQU ((_A2 AND 03FCH) SHR 2)
_BBR3 EQU (((_A3 AND 03H) SHL 6) OR _B3)
_BR3 EQU ((_A3 AND 03FCH) SHR 2)
_BBR4 EQU (((_A4 AND 03H) SHL 6) OR _B4)
_BR4 EQU ((_A4 AND 03FCH) SHR 2)
```

```
LD A, LOW(_BBR1) ;MMUレジスタへの書き込み
OUT (BBR1), A
LD A, LOW(_BR1)
OUT (BR1), A
LD A, LOW(_BBR2)
OUT (BBR2), A
LD A, LOW(_BR2)
OUT (BR2), A
LD A, LOW(_BBR3)
OUT (BBR3), A
LD A, LOW(_BR3)
OUT (BR3), A
LD A, LOW(_BBR4)
OUT (BBR4), A
LD A, LOW(_BR4) ;書き込んでも不変
OUT (BR4), A
```

エイト・サイクルを0にすることもできます。またリセット直後は外部メモリ・アクセスに1ウェイト挿入されていますから、アクセス・タイム150nsのメモリでも使えます。

## USART の使い方

### ● 8251 互換の SIO

シリアル・コントローラは8251とソフトウェア互換のものが内蔵されているので、8251用のプログラムがそのまま使えます。

また USART の TxD, RxD(データ送受信)は専用端子なので、兼用端子の選択設定の必要はありません。

## 〈リスト6〉ノン・ウェイト設定プログラム

```
***** WAITコントローラ初期化 *****
LD A, 0C0H ;NO WAIT
OUT (3BH), A ;SET TO SCR1
```

## 〈リスト7〉SIO 送受信クロックの初期化の例

\*\*\*\*\* システム・レジスタ初期化 \*\*\*\*\*

```
SCRO_INIT: ;DTR, DSR, RTS, /CTS を使う
LD A, 00000010B ;5線インターフェースの場合は
OUT (3AH), A ;このコメントを有効にする
; タイマ・カウンタB1(クロック・ジェネレータ)初期化
TMRB1_INIT:
LD A, 00000110B ;OUTP=0, 連続発生モード, 4分周(GATEなし)
OUT (23H), A ;TCB1Cへのセット
LD HL, 00007H ;時間定数のロード
LD A, L ;
OUT (22H), A ;下位定数のTCB1へのセット
LD A, H ;
OUT (22H), A ;上位定数のTCB1へのセット
```

またリセット直後はタイマ・カウンタBのチャンネル1が、内部でUSARTのTx/C, Rx/C(送受信クロック入力)に接続されているため、そのまま送受信クロックに使用することができます。

リスト7に送受信クロックの初期化プログラムの例を示します。

### ● カウンタの注意点

タイマ・カウンタの定数は実際のカウント値-1とする点が、Z80CTCとは異なります。また兼用端子にはRTSやDTR, CTSがありますが、CTSについては内部でプルダウンされているらしく、3線式のインターフェースでも問題ないようです。

## パラレル・ポートの使い方

### ● パラレル・ポートの設定

汎用パラレル・ポートは、8ビット全5ポートで40ビット分あり、チャンネルAは8ビット2ポート、チャンネルBは8ビット3ポートです。

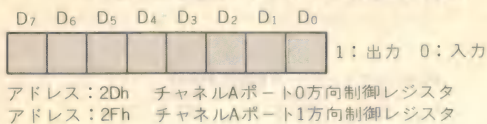
チャンネルAはビットごとの入出力の指定が可能で、チャンネルBは上位下位4ビットごとの入出力や、ビットのセット・リセットが可能です。

またこれらのポートには出力レジスタがあり、入力方向になっていても書き込み値をラッチし、出力方向に切り替えたときにはその値を出力します。

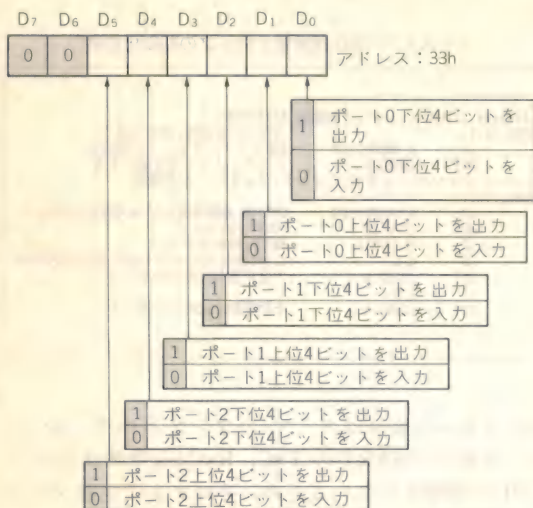
Z80PIOのようなハンドシェイクの制御線はありません。よって入出力時に割り込みを発生させることはできませんが、後述の割り込みコントローラと組み合わせることで同様のことをさせることも可能でしょう。

リセット直後はすべて入力方向に、また内部の出力レジスタも0に初期化されています。図15にチャネ

〈図 15〉 パラレル・ポート方向制御レジスタ



(a) チャンネルAの入出力方向



(b) チャンネルBの入出力方向

ル A, チャンネル B の方向制御レジスタの構成を示します。

リセット直後なので兼用ピンの状態はパラレル・ポートが選択されています。よって兼用端子の選択設定なしで、すぐに入力することができます。またチャンネル B ポート 0 だけは専用端子となっていて、兼用端子の設定の必要はありません。

#### ● 出力に設定した例

リスト 8 にチャンネル B ポート 0 への出力例を示します。チャンネル B の方向はパラレル・ポート B 方向制御レジスタ(アドレス 33h)の下位 6 ビットで指定します。

リスト 9 にチャンネル B ポート 1 への出力例を示します。入力状態でデータ・レジスタに書いたデータは方向を出力に指定しても“0”に初期化されるわけではなく、そのまま出力されます。したがって、入力方向で読み出し→入力方向で書き込み→出力方向に切り替え→次のデータの書き込みの手順で、ブルアップ/ブルダウんにかかわらず、ノイズを出すことなく出力に切り替えることができます。

リスト 10 にチャンネル A ポート 0 のビット 0 のみの出力例を示します。またリスト 11 にチャンネル B ポート 0 のビットごとの初期化例を示します。ここではビット 0 のみノイズを出さずに出力に設定した後、ビット操作機能を用いトグルさせています。

〈リスト 8〉 チャンネル B ポート 0 の出力の例

```
***** PPB0出力テスト *****
LD A,00000011B ;PPB0[7:0]=出力
OUT (33H),A
LD A,33H ;出力データ DATA
MAIN: OUT (33H),A
      CPL
      JR MAIN
```

〈リスト 9〉 チャンネル B ポート 1 への出力の例(初期化時にノイズを出さない)

```
*** PPB0出力テスト(ノイズを出さない方法) ***
IN A,(31H) ;PPB1入力
OUT (31H),A ;出力レジスタへのリセット
LD A,00001100B ;PPB1[7:0]=OUTPUT
OUT (33H),A

MAIN: LD A,55H ;OUTPUT DATA
      OUT (31H),A
      CPL
      JR MAIN
```

〈リスト 10〉 チャンネル A ポート 0 ビット 0 の出力の例

```
***** PPA0出力テスト *****
LD A,01H ;PPA0[0]=出力
OUT (2DH),A

MAIN: LD A,01H ;出力 DATA
      OUT (2CH),A
      XOR 01H
      JR MAIN
```

〈リスト 11〉 チャンネル B ポート 0 のビット単位の出力の例

```
*** PPB0ビット単位出力テスト(ノイズを出さない方法)
IN A,(30H) ;PPB0入力
OUT (30H),A ;出力レジスタへのリセット
LD A,00000001B ;PPB0[3:0]=出力
OUT (33H),A

MAIN: LD A,11000001B ;PPB0[0]=1
      OUT (33H),A ;DO ビット・セット
      LD A,11000000B ;PPB0[0]=0
      OUT (33H),A ;DO ビット・リセット
      JR MAIN
```

## タイマ・カウンタの使用例

### ● タイマ・カウンタ A の使い方

タイマ・カウンタ A は、

- ・分周モード
- ・PWM モード
- ・パルスモード
- ・パルス幅/周期測定モード

の四つの動作モードをもち、外部クロックやシステム・クロックの選択、トリガのエッジなどの組み合わせを含めると、Z80CTC などとは比較にならないほど高性能な 16 ビット・タイマ・カウンタです。チャンネル 0 と 1 の 2 チャンネルを内蔵しています。



## 〈リスト12〉 タイマ・カウンタAのシステム・クロックの分周の例

```

;***** TMRAチャンネル0 システム・クロック分周テスト *****
IN      A, (3AH)      ;現在のSCRO設定値の読み込み
SET     4, A          ;OUTBS2, OUTBPO, OUTA1, OUTA0に
OUT     (3AH), A      ;97~100ビットの機能を切り替え

LD      A, 00000001B   ;システム・クロックを分周して1出力
OUT     (28H), A      ;タイマ・カウンタAチャネル0のレジスタへセット
LD      HL, 0004H      ;分周数=4+1
LD      A, L           ;下位設定
OUT     (28H), A
LD      A, H           ;上位設定
OUT     (28H), A

```

## 〈リスト13〉 タイマ・カウンタAのPWMモードの例

```

;***** TMRAチャンネル0 PWMテスト *****
IN      A, (3AH)      ;現在のSCRO設定値の読み込み
SET     4, A          ;OUTBS2, OUTBPO, OUTA1, OUTA0に
OUT     (3AH), A      ;97~100ビットの機能を切り替え

LD      A, 00001111B   ;ビット0はシステム・クロックの212のPWMモード
OUT     (28H), A      ;タイマ・カウンタAチャネル0のレジスタへセット
LD      HL, 333H      ;パルス幅=333H+1=820
LD      A, L           ;下位設定
OUT     (28H), A
LD      A, H           ;上位設定
OUT     (28H), A

```

### ● 分周器の例

もっとも単純にシステム・クロックを分周する例をリスト12に示します。

最初のシステム制御レジスタSCR<sub>0</sub>へのOUT命令で、端子機能をタイマ・カウンタに切り替えます。ここでは、タイマ・カウンタAチャンネル0を使ってシステム・クロックを5分周して、OUTA<sub>0</sub>(OUT端子)をトグルさせているので、システム・クロックの1/10のクロック出力が得られます。16ビットの定数を2回に分けて設定する必要があるため、下位・上位のライト・シーケンスにしたがったコマンド書き込みが必要です。

GATEA<sub>0</sub>(ゲート入力)は、システム制御レジスタで未使用のままに設定しているため、内部で常に1(カウント・イネーブル状態)が入力されています。

### ● PWMモードの例

同様にシステム・クロックを使ったPWMモードの例をリスト13に示します。ここでは、タイマ・カウンタAチャンネル0を使ってシステム・クロックの2<sup>12</sup>+1=4097クロック周期で、333h+1=820クロック時間をHレベルとしました。積分回路(時定数10ms程度)を通すと約1Vが得られ、簡易的ですがD-Aコンバータにもなります。

### ● パルス幅カウントの例

GATE端子に入力される信号の、パルス幅を測定する例をリスト14に示します。実験的に、HC4040を使ってXCLK<sub>0</sub>に1.2288MHz、GATEA<sub>0</sub>にその1/256の4800Hzを入力してみました。

SCR<sub>0</sub>で、GATEA<sub>0</sub>(入力として使いますが、切り替

## 〈リスト14〉 タイマ・カウンタAのパルス幅カウントの例

```

;***** TMRAチャンネル0 : パルス幅測定テスト *****
IN      A, (SCRO)      ;現在のSCRO設定値の読み込み
SET     4, A          ;OUTBS2, OUTBPO, OUTA1, OUTA0と
SET     5, A          ;GATEA0に
OUT     (SCRO), A      ;79, 97~100ビットの機能を切り替え

LD      A, 00100100B   ;GATEA0の立ち上がりから立ち下がり
OUT     (TCA0C), A     ;までを、XCLKで1回測定

LOOP:   IN      A, (TCA0C) ;ステータス・リード
        BIT     7, A      ;bit7 チェック
        JR      Z, LOOP   ;測定未終了ならLOOPへ

LD      A, 00111000B   ;カウンタ・ラッチ・コマンド
OUT     (TCA0C), A

IN      A, (TCA0)      ;下位カウント値をレジスタに
LD      L, A
IN      A, (TCA0)      ;上位カウント値をレジスタに
LD      H, A
IN      A, (TCA0)      ;下位カウンタ値をレジスタに
LD      L, A
IN      A, (TCA0)      ;上位カウンタ値をレジスタに
LD      H, A

```

えが必要)を使用する(イネーブル)に切り替え、測定を立ち上がりでスタートし、立ち下がりで終了するよう設定しました。TCA<sub>0</sub>の1, 2回目は0FFFFhからダウン・カウントしたカウント値であり、3, 4回目がその補数でカウント値となります。

測定結果のHLレジスタの内容は、正しい0080h(128)の場合と007Fh(127)の場合がありました。周波数を変えてもみましたが、タイミングによって±1の誤差は出るようです。

なお、外部カウント基準クロックXCLK<sub>0</sub>/XCLK<sub>1</sub>に関しては入力端子なので兼用端子の切り替えは不要です。しかし端子をポート出力に使った場合は、XCLKを使用するとポート出力がクロックとなるようです。

この他、パルス・モードや2チャンネルをシリーズにした使い方もできます。

### ● タイマ・カウンタBの使い方

タイマ・カウンタBは、

- ・ PWM モード
- ・ WDT モード
- ・ 連続カウント・モード
- ・ 単発カウント・モード

の四つの動作モードをもつ高機能な16ビット・タイマ・カウンタです。チャンネル0~2の3チャンネルを内蔵しています。

### ● 分周器の例

もっとも単純にシステム・クロックを分周する例を、リスト15に示します。動作波形を図16に示します。

タイマ・カウンタBチャンネル0を使って、システム・クロックの4分周をGATE機能なしで連続カウントします。ここではカウント初期値を3にしたので、4×(3+1)=16分周ごとにOUTBP<sub>0</sub>端子はトグル動作します。OUTBS<sub>0</sub>端子には4システム・クロックが

## 〈リスト 15〉 タイマ・カウンタ B の分周器の例

```

;***** TMR Bチャンネル0 : 連続カウント・テスト *****
IN      A, (3AH)      ;現在のSCRO設定値の読み込み
SET     3.A           ;OUTBS0, OUTBP1, OUTBP2, SYNCと
SET     4.A           ;OUTBS2, OUTBP0, OUTA1, OUTA0に
OUT     (3AH), A       ;1, 4, 5, 6, 97~100の機能を切り替え

LD      A, 00000110B   ;4分周ゲート機能なし、連続カウント
OUT     (21H), A
IN      A, (21H)
LD      HL, 3          ;ステータス・リード(シーケンス・クリア)
                     ;3カウント初期値=3
                     ;OUTBP0=9.8304MHz/(4*(3+1)*2)=307.2KHz
                     ;下位書き込み
LD      A, L           ;下位書き込み
OUT     (21H), A
LD      A, H           ;上位書き込み
OUT     (21H), A

```

## 〈リスト 16〉 タイマ・カウンタの PWM 出力の例

```

;***** TMR Bチャンネル0 PWMテスト *****
IN      A, (3AH)      ;現在のSCRO設定値の読み込み
SET     3.A           ;OUTBS0, OUTBP1, OUTBP2, SYNCと
SET     4.A           ;OUTBS2, OUTBP0, OUTA1, OUTA0に
OUT     (3AH), A       ;1, 4, 5, 6, 97~100の機能を切り替え

LD      A, 00001111B   ;4分周ゲート機能あり、PWMモード
OUT     (21H), A
IN      A, (21H)       ;ステータス・リード(シーケンス・クリア)
LD      A, 7           ;パルス周期=7+1
OUT     (21H), A
LD      A, 2           ;パルス幅=2+1
OUT     (21H), A

```

〈図 16〉 タイマ・カウンタ B の分周出力波形の例



カウント値 0 のたびに出力されます。ステータスのリードはライト・シーケンス・クリアのために実行します。

### ● PWM モードの例

リスト 16 に PWM モードの例を示します。動作波形を図 17 に示します。

タイマ・カウンタ B チャンネル 0 を使い、システム・クロックを 4 分周した信号を基準にし、GATEB<sub>0</sub> が H レベルの期間に、8 クロック周期で 3 クロック分の H レベルを出力し、これを繰り返します。これが PWM 出力です。

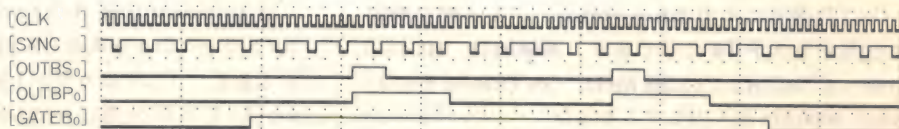
GATEB<sub>0</sub> は入力のため端子機能の切り替えは必要ありません。GATEB<sub>0</sub> = “1” の期間は正しく PWM 波形が出力されますが、GATEB<sub>0</sub> = “0” の期間ではカウント機能が停止します。図 16 の動作波形では、丁度 2 パルス分に GATEB<sub>0</sub> 期間がありますが、GATEB<sub>0</sub> が OUTBP<sub>0</sub> = “1” の期間中に “0” になると、波形だけでは正しい PWM といえなくなります。このほかにも、単発カウント・モードや、WDT モード、システム・クロックの 256 分周までのプリスケアラの設定などあります。

## 割り込みコントローラの使用例

### ● Z80 モード 2 対応

割り込みコントローラは、Z80 のモード 2 割り込み

〈図 17〉  
タイマ・カウンタ B の  
PWM 出力波形の例



をサポートしています。割り込み要因としては、

- ・内蔵タイマ・カウンタからの 5 要因
- ・内蔵 USART からの 3 要因
- ・外部割り込みリクエスト (IR<sub>7</sub>~IR<sub>0</sub>) ピンからの 8 要因 (IR<sub>7</sub>~IR<sub>0</sub> は、パラレルポート A チャンネル 0 と兼用)

の 16 個がサポートされています。割り込みの優先度や許可、ベクタなどを制御するレジスタが 6 個あります。

高性能なので比較的簡単な外部割り込みリクエスト・ピンからの割り込み例をリスト 17 に示し、簡単な解説をします。

### ● 外部割り込み

リスト 17 では、最初に各種レジスタの設定を行います。レベル/エッジ・レジスタ、ベクタ・レジスタ、マスク・レジスタの設定を行っています。この例では割り込み番号 IR<sub>3</sub>~IR<sub>0</sub> を許可としました。

プログラム例は実験的に外部から 40 k, 20 k, 10 k, 5 kHz の順に、IR<sub>0</sub>, IR<sub>1</sub>, IR<sub>2</sub>, IR<sub>3</sub> へ HC4040 の出力を与え、エッジ・モード(立ち上がり)で割り込みを受けた回数をワーク・エリアに書き込みます。IR<sub>3</sub> 受け付け終了時に割り込みディセーブル状態でメイン・ルーチンに戻り値を確認します。

グループ・レジスタを設定していないので、割り込み優先度はデフォルトの IR<sub>3</sub>~IR<sub>0</sub> の順になり、値はそれぞれ 4, 2, 1, 1 となり、正しい値となりました。

多重割り込みの優先度に関しては Z80 ファミリの IEI, IEO のデジィ・チェーン結線や信号のディレイなどを気にすることなくプログラムで設定できます。

このほか、タイマ・カウンタや USART からの割り込み、IR<sub>7</sub>~IR<sub>0</sub> 端子をポート出力として切り替える、



# 〈リスト 17〉 割り込みプログラムの例

;***** 割り込みコントローラテスト I Rピン多重割り込み *****		
LD	SP, 9000H	
DI	2	;モード2使用
;	LERH/L初期化、リセット後はレベル・モード	
LD	A, 0FH	;IR[7:4]レベル, IR[3:0]エッジ
OUT	(34H), A	
LD	A, 0	;IR[15:8]レベル
OUT	(35H), A	
;	IVR初期化	
LD	HL, INTVEC	
LD	A, H	
LD	I, A	;上位ベクタをレジスタにセット
LD	A, L	
OUT	(37H), A	;下位ベクタをIVRにセット
;	IMR初期化	
LD	A, 0FH	;IR[7:4]不可, IR[3:0]許可
OUT	(36H), A	
LD	A, 0FFH	;IR[15:8]不可
OUT	(37H), A	
CHECK:	IN A, (2CH)	;IR3 (PPA0[3])が
BIT	3, A	;1なら待つ
JR	NZ, CHECK	
LOOP:	JR \$	;割り込み可
		;無限ループ
INTIRO:	EI	;EI命令を実行しないと
LD	A, (IRO_CNT)	;多重割り込みできない
INC	A	
LD	(IRO_CNT), A	
EI		
RETI		
INTIR1:	EI	
PUSH	AF	
LD	A, (IR1_CNT)	
INC	A	
LD	(IR1_CNT), A	
POP	AF	
EI		
RETI		
INTIR2:	EI	
PUSH	AF	
LD	A, (IR2_CNT)	
INC	A	
LD	(IR2_CNT), A	
POP	AF	
EI		
RETI		
INTIR3:		;最高優先度のためEI不要
PUSH	AF	
LD	A, (IR3_CNT)	
INC	A	
LD	(IR3_CNT), A	
POP	AF	
EI		;リターンするだけで次の割り込みは
RETI		;受け付けない
INTVEC:	ORG (\$+1FH) AND 0FF0H	
DW	INTIRO	:IRO
DW	INTIR1	:IR1
DW	INTIR2	:IR2
DW	INTIR3	:IR3
DW	0	:IR4
DW	0	:IR5
DW	0	:IR6
DW	0	:IR7
DW	0	:IR8
DW	0	:IR9
DW	0	:IR10
DW	0	:IR11
DW	0	:IR12
DW	0	:IR13
DW	0	:IR14
DW	0	:IR15
ORG	8000H	;RAM領域
IRO_CNT:	DB 0	
IR1_CNT:	DB 0	
IR2_CNT:	DB 0	
IR3_CNT:	DB 0	

ソフトウェア割り込みなどができます。

## ● 最後に

最後にリスト 18 に KL5C8012 の内蔵 I/O ポートのアドレスを定義したファイルを示します。ヘッダ・ファイルとして活用してください。

ここまで高機能の CPU なら、TMPZ84C015 のような HALT モードでの低消費電力化や、外部 I/O をアクセスするためのプログラマブル・チップ・セレクト端子などが内蔵されると、さらに使いやすくなるような気がします。

〈北 孝〉

## ●参考・引用文献

- (1) 8ビットマイクロプロセッサ TLCS-Z80ASSP, 1991年, 朝東芝.
  - (2) VOLUME 1 DATABOOK, MICROPROCESSORS AND PERIPHERALS, ZILOG.
  - (3) Z80 Family User's Manual, ZILOG.
  - (4)\*高速 8ビットマイクロコントローラ KL5C8012 ハードウェアマニュアル, 川崎製鉄㈱.
  - (5) 高速 8ビットマイクロコントローラ KL5C8012 アプリケーションノート, 川崎製鉄㈱.
- (トランジスタ技術 1994 年 6 月号および 10 月号に加筆, 修正)

# 〈リスト 18〉 KL5C8012 の内蔵 I/O アドレスのヘッダ・ファイルの例

;*****		
; KL5C8012 : INTERNAL I/O REGISTER ADDRESS DEFINITION		
;*****		
BBR1	EQU 00H	;mmu Border/Base Reg 1
BR1	EQU 01H	; Base Reg 1
BBR2	EQU 02H	; Border/Base Reg 2
BR2	EQU 03H	; Base Reg 2
BBR3	EQU 04H	; Border/Base Reg 2
BR3	EQU 05H	; Base Reg 3
BBR4	EQU 06H	; Border/Base Reg 4
BR4	EQU 07H	; Base Reg 4
TCB0	EQU 20H	;Timer/Counter B-ch0 count reg
TCB0C	EQU 21H	; Control reg
TCB1	EQU 22H	;Timer/Counter B-ch1 count reg
TCB1C	EQU 23H	; Control reg
TCB2	EQU 24H	;Timer/Counter B-ch2 count reg
TCB2C	EQU 25H	; Control reg
TCA0	EQU 28H	;Timer/Counter A-ch0 count reg
TCA0C	EQU 29H	; Control reg
TCA1	EQU 2AH	;Timer/Counter A-ch1 count reg
TCA1C	EQU 2BH	; Control reg
PPA0	EQU 2CH	;Parallel Port A-ch0 data reg
PPA0D	EQU 2DH	; Direction reg
PPA1	EQU 2EH	; Port A-ch1 data reg
PPA1D	EQU 2FH	; Direction reg
PPB0	EQU 30H	;Parallel Port B-ch0 data reg
PPB1	EQU 31H	; Port B-ch1 data reg
PPB2	EQU 32H	; Port B-ch2 data reg
PPBD	EQU 33H	;Parallel Port B-Direction reg
LERL	EQU 34H	;Level/Edge Reg-Low for write
LERH	EQU 35H	;Level/Edge Reg-High for write
PGRL	EQU 34H	;Priority Group Reg-Low for write
PGRH	EQU 35H	;Priority Group Reg-High for write
ISRL	EQU 34H	;In Service Reg-Low for read
ISRH	EQU 35H	;In Service Reg-High for read
IMRL	EQU 36H	;Interrupt Mask Reg-Low for write/read
IMRH	EQU 37H	;Interrupt Mask Reg-High for write/read
IVR	EQU 37H	;Interrupt Vector Reg for write
SIO	EQU 38H	;Serial I/O data reg
SIOC	EQU 39H	;Serial I/O Control reg
SCR0	EQU 3AH	;System Control Reg0
SCR1	EQU 3BH	; Reg1

アセンブラによるソフト開発からICEによるデバッグまで

## Z80マイコン・システム開発手順

野口智樹/藤丘勝信

### Z80 マイコン・システムの開発手順

#### マイコンが理解できる言葉 機械語

##### ● 機械語とは何か

皆さんご存じのとおり、マイコン(CPU)はプログラムがないとまったく動作してくれません。マイコンは一つ一つの命令を解読し、それを忠実に実行するだけです。

さて、そのマイコンが理解してくれる命令とはどんな命令なのでしょう。もうすでにおわかりだと思いますが、マイコンが理解できる命令とは、人間が使っている言葉のようなものではなく、01hや3Ehといったような、数値で表されたものです。

このような、マイコンが直接に理解できる数値による命令が機械語と呼ばれるものです。

命令といっても人間から見れば単なる数字の羅列です。人間が機械語を読んでみてもすぐに理解できるものではありませんし、またその必要もほとんどありません。人間がマイコンのプログラムを作成するために、もっと簡単に理解できるように、機械語と1対1に対応させて表現した言語がアセンブリ言語です(図1)。

##### ● アセンブラとは

大むかし(といっても十数年前)までは、アセンブリ言語から機械語への変換作業は人間が1行ずつ手作業

で行っていました。現在ではこのような変換はソフトウェアが自動的に行ってくれます。このようなソフトウェアのことをアセンブラといいます。

また今のアセンブラでは、単なる機械語への変換だけではなく、プログラム開発が簡単になるようにいろいろと便利な機能が備わっています。これらの機能についてはここでは省きます。

##### ● アセンブリ言語のメリット

世の中にはさまざまなプログラム言語がありますが、その中でもっともCPUに近い部分のプログラミングはアセンブリ言語によって行われています。

CPUに近い部分というのは漠然とした言い方かもしれませんが、アセンブリ言語は機械語と1対1で対応している言語ですから、CPUの動作の一手一投足までをコントロールすることができるということです(図2)。

つまりアセンブリ言語を使えば、CPUのもつパフォーマンスを100%引き出すことができます。

##### ● アセンブリ言語のデメリット

アセンブリ言語には当然デメリットもあります。高級言語(C言語やFORTRAN, BASICなど)に比べ、一般的にソフトウェアが大規模になるほど開発効率が低下するなどといった問題点もあります(アセンブラのほうが高級言語より得意だ! という人もいるかも

〈図1〉  
機械語とアセンブリ言語の対応

機械語	マイコンの処理	アセンブリ言語
3E 01	Aレジスタに1を代入	LD A, 1
21 34 12	HLレジスタに1234hを代入	LD HL, 1234h
3C	Aレジスタに1を加算	INC A
FE 02	Aレジスタと2を比較	CP 2
CA 78 56	ゼロ・フラグが1ならば5678hへの分岐	JP Z, 5678h

↑  
数字の羅列  
では人間は  
わからない、

↑  
これなら人間にわかりやすいがいちいち  
こんな文章で書くのはめんどろ、

↑  
人間に意味がわかる  
程度にまで記号化し  
てマイコンのプログ  
ラムを書く、



〈図2〉 アセンブリ言語はCPUを細くコントロールできる

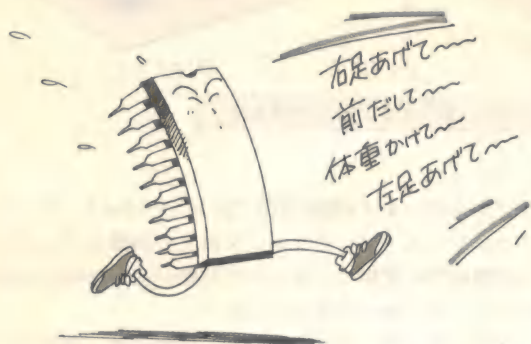
LD A, 12H	Aレジスタに12hを代入
CP B	Bレジスタと比較
JP Z, JMP1	ゼロ・フラグが1ならJMP1へジャンプ

CPU動作の一つ一つを細かく指定できる

〈図3〉 高級言語は一つの命令でアセンブリ言語の数十行分に当たる

```
if (A > B) printf("Hello!");
else printf("CQ!");
```

一つの命令でアセンブリ言語の数十行にあたる



しませんが…).

なぜアセンブリ言語では開発効率が低下するのでしょうか。例えば小数点演算などがある場合、アセンブラでは数十行ものプログラムになりますが、高級言語では1行で済んでしまうからです。

このようにアセンブリ言語では、その一つ一つの命令が非常に簡単なことしかできないので、何か意味のある動作をさせるためにはいくつもの命令を組み合わせなければなりません。ところが高級言語は1行の命令だけでも意味のある動作をさせることが可能なのです。このことから高級言語に対する言葉として、アセンブリ言語のことを低級言語と呼ぶこともあります(図3)。

以上のことから、プログラムの処理速度やメモリ容量にかなりの余裕があるならば、ほとんどの部分に高級言語を使い、高速な処理が必要な部分だけをアセンブリ言語でプログラミングすれば、バランスの良いシ

ステムが作れるのです。

## マイコン・システムの開発手順

Z80は一般に小型の組み込み用機器などで多く使用されているマイコンですが、このような組み込み機器を目的とする場合のソフトウェア開発は、どのような手順で行えばよいのでしょうか。

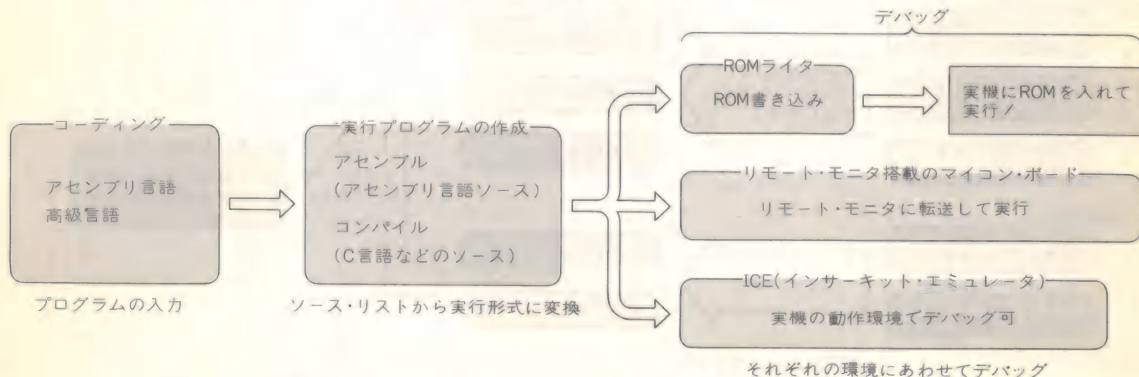
基本的な開発手順は図4のようになります。

### ● 開発環境の整備

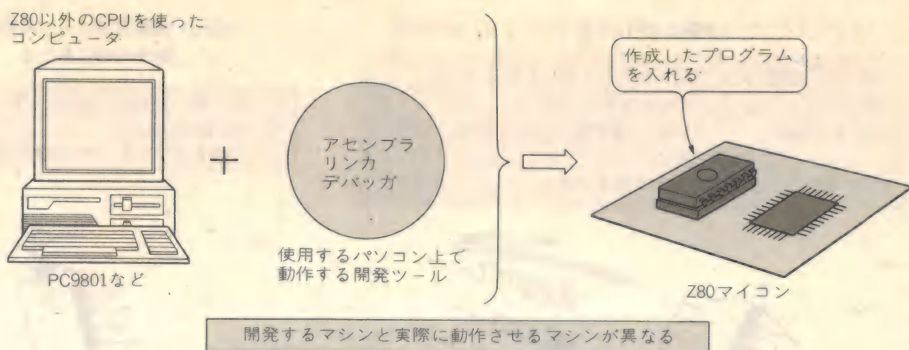
マイコンの開発環境としては、現在では一般にパソコンが使われているようです。この場合、当然アセンブラなどは開発用のパソコン上で動作するものが必要となります。また、アセンブラ以外にもコーディングを行うためのエディタ、ROM書き込みのためのソフトウェアや書き込み器なども必要になります。

例えばパソコンとしてPC-9801を使う場合は、OS

〈図4〉 プログラムの開発手順



〈図5〉  
クロス開発とは



であるMS-DOS上で動作するZ80用のアセンブラ、そしてMIFESやVzなどのテキスト・エディタをそろえます。

また、PC-9801で使われているCPUはZ80とはまったく別のCPUです。このように、あるCPUのアセンブル作業を別のCPUで動作するプログラムによって行うことをクロス開発などと呼び(図5)、このとき使用するアセンブラを、クロス・アセンブラと呼びます。

### ● コーディング

コーディングとはソース・プログラムを作る作業のことです。つまりLD A, (1234H)というようにアセンブリ言語を記述していく作業です。

プログラムを入力するためにはワープロなども使えますが、一般にはさきほど説明したようなエディタと呼ばれるソフトウェアを使います。

### ● アセンブル

コーディングが終わると、アセンブリ言語で書かれたソース・プログラムを機械語に変換するためにアセ

ンブルを行います(高級言語ではコンパイル)。アセンブラはソース・プログラムに文法上の問題などがなければ機械語に変換し、何らかの問題があった場合には、エラー・メッセージを出力します。

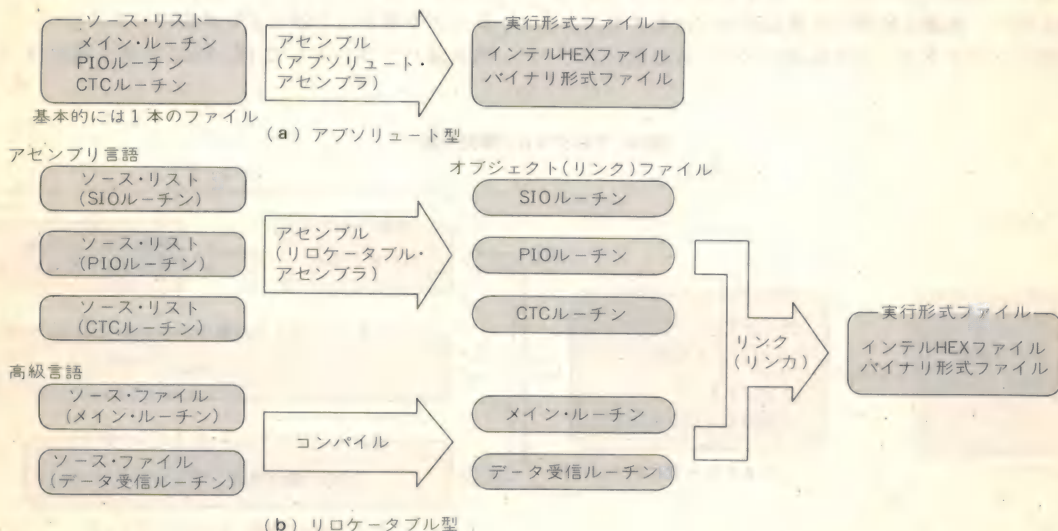
文法上のエラーなどが発生した場合には、再度コーディングに戻り、エディタで間違いを訂正してアセンブルします。

またアセンブラには、アブソリュート型とリロケータブル型とがあり、場合によって使い分けします(図6)。

アブソリュート型とは、ソース・リストから直接実行形式ファイルを出力するタイプのアセンブラです。開発するプログラムの規模が小さい場合はこれで十分でしょう。

リロケータブル型とは大規模プログラム開発に適しています。ソース・リストを機能ごとに分割し、いくつかのファイルからオブジェクト・ファイルと呼ばれる中間形式のデータを作成します。そして必要なオブジェクト・ファイルを、リンカと呼ばれるツールで結

〈図6〉 アブソリュート型とリロケータブル型の違い





合し、実際に動作できる実行形式ファイルを作成する  
ものです。またC言語などの高級言語と合わせてプ  
ログラムを開発するには、リロケートブル型アセンブ  
ラを使います。

### ● デバッグ

作成した実行プログラムを実際に動作させて、不具  
合がないかどうかチェックする行程です。

この行程は使用するツールによって作業の流れがい  
ろいろあります。いちばん基本的なやり方は、ROM  
ライターでプログラムを書き込み、実機のZ80マイコ  
ンのROMソケットに挿入して動作させる方法です。

二つめはZ80マイコン・ボードにリモート・モニタ  
を搭載して、パソコンと通信しながらデバッグする手  
法です。

三つめはICE(インサーキット・エミュレータ)を使  
う方法で、レジスタの値の変化やプログラムのステッ  
プ・実行などが可能です。

### ● プログラムが動かないわけ

さて、プログラムが正常に動かないのならなぜアセ  
ンブラがエラーを出さなかったのでしょうか？

アセンブラとは、文法や命令のスペル・ミス、未定  
義の名前や範囲外の値が代入されたときなどしかエラ

ーを出力しません。

つまり、プログラムの論理的なミスは人工知能でも  
搭載しないかぎりチェックは不可能なのです。

また、パソコンでのプログラミングとマイコンでの  
プログラミングの大きな違いとして、パソコンでは  
(本来なら)ハードウェアは完全に動作することがわか  
っていますが、マイコンでは、プログラムの間違いに  
よるものではなく、ハードウェアの間違いの可能性も  
考えられる点です。

どちらにしろ、プログラムが動かなければ、どこに  
原因があるかを突き止めるデバッグという作業に入り  
ます。

### ● デバッグとは

ROMに書かれた機械語プログラムは、電源投入直  
後から一気に実行されるので、動かない原因を探るた  
めに人間が「あ、いまAレジスタに10hが代入され  
たなあ」などとやってる暇はありません。

そこで、デバッグ作業を効率よく行うために、一般  
的にはデバッグというツールを使います。

デバッグにはいろいろな種類がありますが、デバッ  
グ手法については後でまとめて解説します。

## アセンブラによるソフトウェア開発

すでに説明したように、マイコンのソフトウェア開  
発にはアセンブラや高級言語が使われます。ここでは、  
いちばん基本的なアセンブラによるソフトウェア開発  
について解説します。

### Z80 アセンブラの基礎

#### ● アセンブリ言語の構文

アセンブリ言語には文法や記述における作法、そし  
て必ずそうしなければならないと決まっているわけ  
ではないが、慣習的にそうしているといったことなど  
があります。

図7にアセンブリ言語の構文を示します。アセン

〈図7〉アセンブリ言語の構文

ラベル	ニモニック	オペランド	コメント	行末まで
LBL1:	LD	HL, 8000H	;Buffer Address	
	IN	A, (80H)	;Data Input	
	CP	2	;Ready?	
	JP	NZ, LBL1	;Z=0 : Busy	

TABキーでカラムをそろえる

ブリ言語はラベル、ニモニック、オペランド、そして  
コメントで1行が構成されています。

#### ▶ ラベル(Label)

今までの例ではアドレスを指定するのに直接1234h  
などとアドレス値を指定しましたが、機械語が数字の  
羅列でわかりづらいのと同様、そのアドレスは何のデ  
ータを保存しておくアドレスなのか直感的にわかりま  
せん。

例えば、Aレジスタに外部スイッチの状態が入って  
いて、これを保存しておくメモリであれば、

LD (8005H), A

と書くよりは、

LD (SWSAVE), A

と書いたほうが、スイッチの状態をセーブしておくア  
ドレスなんだということがわかります。

このSWSAVEがラベルであり、プログラムの各  
所で参照される場合に命名する、アドレスの名前のこ  
とです。

ラベルは一般的には第1カラム(1行の最左端)から  
書くことになっていますが、多くのアセンブラでは必  
ずしもその必要はありません。ただし、ニモニックよ  
りも左側に書かなければなりません。

通常、ラベルの最後には“:”(コロン)を付加しま

す。ラベルとして使用可能な文字は、英数字(A～Z, a～z, 0～9)と“\_”に“@”などですが、先頭の文字だけは数字であってはなりません(ラベルか値かの判別がつかなくなる場合がある)。

ニモニックを書き始める行まではTABキーで移動するのが一般的なので、“:”とニモニックの間に1文字空白を入れると、ラベルに使える文字の長さは6文字となります。このことからラベルの文字数は伝統的には6文字以内で書くことになっていますが、現在では、多くのアセンブラが10文字以上の長いラベル名でも認識するようになっています。

長さにとらわれず、直感的にわかりやすいラベル名をつけたほうがよいでしょう。

#### ▶ ニモニック(Mnemonic)

ニモニックとはアセンブラの命令のことで、Z80の場合LD A, Bの例でいえばLDの部分のことで、

#### ▶ オペランド(Operand)

オペランドとはニモニックなどの命令に続くパラメータのことです。この書式はニモニックによって決まります。

#### ▶ コメント(Comment)

1行の中で“;”(セミコロン)があると、それより右側の文字列はすべてコメントとみなされます。

〈表1〉数値定数の表現

表現	基数	例	
~B	2進数	110B	10111010b
~O	8進数	17O	12o
~Q	8進数	17Q	17q
~D	10進数	100D	100d
~	10進数	100	-123
~H	16進数	1AH	0A4h

コメント部分の使い方は自由です。細かいプログラムの説明や解説などを記入しておくのがよいでしょう。

#### ● 数値定数の表現

数値定数は一般的に表1のように表現します。2進数は“B”を、16進数には“H”をつけます。また16進数で最上位桁がA～Fで始まるときはラベルと区別するために頭に“0”をつけます(例:FFFFHのときは0FFFFH)。

またアセンブラによっては10進数で“D”を、8進数で“O”や“Q”が使えるアセンブラもあるようです。

#### ● 数式を使った値の代入

一般的なアセンブラでは、値の代入に数式も使えます。

例えば、3バイトである意味をもつデータが5組あ

## コメント記述はマクロな目で

よくプログラムの1行ごとに詳細なコメントを記述しているリストを見受けます。それでプログラムが本当に読みやすいのであれば問題ないのですが、かえってゴチャゴチャとわかりにくくなる場合もあります。

また例えばリストAのようなコメントは、ニモニックの動作を日本語にただけのことでほとんど意味がありません(初心者のために、Z80の動作を示す意味なら別ですが)。

よほど複雑なプログラムでない限りは、マクロ(大まか)な見かたでコメントを記述したほうが読みやすい場合が多いと思います。

そのサブルーチンで処理している内容や、例えばCP Bというような、AレジスタとBレジスタの比較も、何と何のデータを比較しているのかなどをコメントに書くべきです。

また実際に仕事でマイコンのプログラムを作る場合、プログラム・リスト以外に、ハードウェアや、通信データのフォーマットなどの細かな仕様書を用意します。

よって、何のデータか忘れてしまいそうな文字列

#### 〈リストA〉意味のないコメント、意味のあるコメント

```
LD      A, B      ;B.regのデータをA.regに代入
ADD     A, C      ;A.regとC.regを足す
```

#### (a) 日本語アセンブラ(?)の意味しかないコメント

```
LD      A, 1011011B ;bit7 割り込み可 bit6 カウンタMode
OUT     (CTCO), A   ;bit2 タイム・コンスタント書き込み
DEFB    "ADC", 13, 10 ;A-D変換開始コマンド
```

#### (b) ハードウェアの仕様書がなくてもわかるコメントの例

データの意味や、ビットごとに処理が割り当てられたポートの設定部分には、簡単なフォーマットやポートのビットの意味をコメントとして記述し、だいたいの意味はリストを見ただけでわかるようになると、見やすいリストであるといえるでしょう。

もっとも、コメントがなくてもスラスラと読めるようなプログラムが書ければ、バグも少なくなって最高なんです...



〈リスト1〉  
HALT プログラム  
のソース・リスト

CSEG.....プログラム領域であることを示す  
ORG 0.....アドレス0000h番地から次の命令やデータを格納する  
HALT.....Z80CPUのための命令  
END.....ソース・リストの終了を示す

〈表2〉アセンブラの疑似命令

疑似命令	書式	機能	使用例
ORG	ORG 式	ロケーション・カウンタの設定	ORG 100H
EQU	ラベル EQU 式	ラベルの値設定	DATA EQU 80H
DEFB	DEFB 式, ...	バイト型データの定義	DEFB 45H, 67H
DEFW	DEFW 式, ...	ワード型データの定義	DEFW 9000H
DEFS	DEFS 式, ...	領域定義	DEFS 1024
IF ELSE ENDIF	IF 式 ELSE ENDIF	条件アセンブル 式が真なら ELSE または ENDIF ま でを, 偽なら ELSE 以降をアセンブル	IF INPUT EQ 1 OUT (PIOA), A ELSE IN A, (PIOA) ENDIF
CSEG	CSEG	プログラム・コード領域の定義	CSEG
DSEG	DSEG	データ領域の定義	DSEG
INCLUDE	INCLUDE“ファイル名”	ファイルの読み込み	INCLUDE“IOADR.H”
END	END	プログラムの終了指定	END

ったとき、そのデータ・ブロックの全バイト数を単純  
に、

```
LD A, 15
```

と指定しても、プログラムに間違いはありません。し  
かしこれを、

```
LD A, 3 * 5
```

と記述することによって、3バイトのデータが5個で  
15 という値が出てきたことを表現できます。

またラベルやシンボルに対しても数式が使えます。  
例えば、DATA というシンボルを 10h としていると  
とき、A レジスタにこの半分の 8 を入れるとします。こ  
のとき、

```
LD A, 8
```

ではなく、

```
LD A, DATA/2
```

とすることもできます。このように記述することで、  
単に A レジスタに 8 を代入しているということだけ  
でなく、DATA というシンボルの半分の値を代入し  
ているのだという意図を表現することができます。

使用できる数式はアセンブラによってさまざまで、  
単純な四則演算しかできないものから、シフト命令や  
論理演算までできるものもあります。

## アセンブラ疑似命令の使い方

### ● 疑似命令とは

リスト1にもっともプログラム・サイズの小さい  
HALT プログラムを示します。HALT 命令とは、

CPU の動作を停止する命令です。CPU の外部から見  
れば、Z80CPU の HALT 出力端子を L レベルにする  
だけのものです。

実行したい本来の内容は HALT 命令の 1 命令だけ  
ですが、ソース・リストには CSEG や ORG, END な  
どの文字列がついています。Z80 の命令書をもても、  
OR ならありますが ORG などという命令はないはず  
です。

このプログラムの 4 行のうち、実際の Z80CPU へ  
の命令は HALT の 1 行だけです。残りの 3 行はアセ  
ンブラに対しての指示であり、これらは機械語にはな  
りません。

このような命令は疑似命令とよばれ、機械語は生成  
されませんがその意味はとても重要なのです。いくら  
Z80 の命令を理解していても、このような疑似命令を  
正しく使わないと思ひ通りのプログラムはできません。

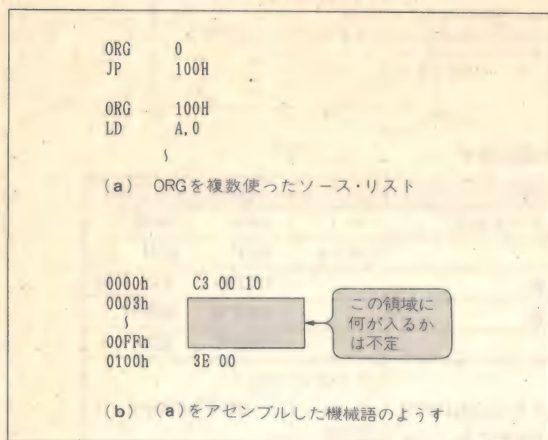
ここでは疑似命令を活用した実践的なアセンブラ・  
プログラムの組み方を説明します。まず基本的な疑似  
命令の効果的な使い方を解説します。

表2に疑似命令の例を示します。

これから解説する疑似命令は、使い方次第で開発効  
率や保守性を向上させることができます。

とくに INCLUDE や IF~ELSE~ENDIF などの疑  
似命令は、上級者ほどこれらの疑似命令を上手く使い、  
プログラムをさまざまな仕様にカスタマイズできるよ  
うな構造にしたり、デバッグの効率を高めたりしてい  
ます。

## ＜リスト2＞ ORGの使用例



### ● ORG

これはアセンブラが生成する機械語やデータを格納するアドレスを指定するものです。厳密にはロケーション・カウンタを設定する疑似命令です。ロケーション・カウンタとは、アセンブラが内部で管理している、現在アセンブル中のアドレスのことです。

Z80のプログラムならば、リセット後0000h番地から実行を開始するので、0000h番地から命令を記述します。

またこの疑似命令は、リスト2の例のように一つのプログラムの中で何度でも使うことができます。

さてこのリスト2の場合、JP命令は0000h番地から格納され、3バイト命令です。ソース・リストでは次の行にまたORG命令が入り、100H番地となっています。よってその次の行のLD A, 0は0100h番地から格納されます。

では、この0003h～00FFh番地の間はどうなるのでしょうか。実はこの領域は通常は不定になります。直接バイナリ・データを出力するアセンブラでは、00hかFFhを埋めるものが多いようです。

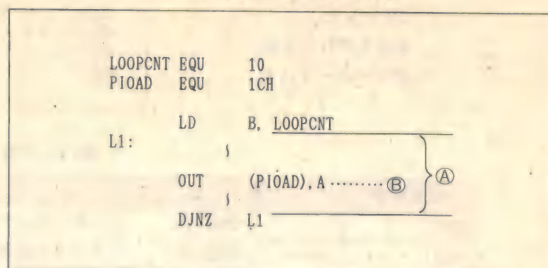
もともとこのような未使用領域は、どんな値が入っているようにプログラムの動作に影響はないはずなので、通常はとくに考える必要はないでしょう。

### ● EQU

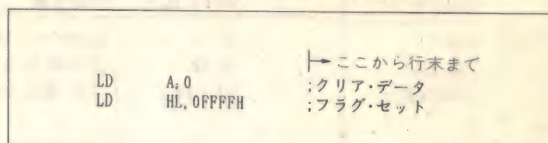
シンボル(定数)定義命令で、おそらくもっとも多く使用される疑似命令でしょう。リスト3に使用例を示します。この例では、Aの部分の処理をLOOPCNT回、つまり10回繰り返します。またBはI/OアドレスPIOAD、つまり1ChにAレジスタのデータを出力することを意味しています。

直接OUT(1CH), Aと記述しても動作はまったく同じです。しかしシンボルで記述することにより、そのアドレスがパラレル・ポートのチャンネルAのデータ・ポートであるということを明示できるわけです。

## ＜リスト3＞ EQUの使用例



## ＜リスト4＞ コメントの使用例



またハードウェアの変更で、I/Oアドレスが移動することも考えられます。もし直接アドレスの値を記述していた場合、ソース・リストのいたるところに散らばっているOUT(1CH), Aという部分のアドレスをすべて書き換える必要があります。

これがシンボルで定義してあれば、ソース・リスト先頭のEQU文を書き換えるだけで済みます。

しかし、シンボルを乱用しすぎるとプログラム・リストを眺めていてもラベルの値を覚えていなければかえって読みにくくなることもあります。

### ● コメント(;)

アセンブラは、コメント(;)より右側の文字を認識しません。したがってシンボルやラベル、ニモニック以外の文字列を記述することができます(リスト4)。ここにはプログラムの説明を書き込むことができます。

### ● DEFB, DEFW

DEFBはバイト型(1バイト)の、DEFWはワード型(Z80の場合2バイト)の定数データを定義します。使用例をリスト5に示します。

またDEFBは文字列の定義も、シングルのクォーツ(あるいはダブル・クォーツ)により区切って表現できます。さらに複数のデータを並べたり文字列の後ろにバイナリ・データを入れたり、シンボルやラベル(アドレスの名前)を記述することもできます。

またアセンブラによっては、文字列の定義にDEFBは使えず、文字列格納専用の疑似命令として、DEFMやDEFCを使う場合もあります。

さらに省略形としてDB, DWが使えるアセンブラもあります。

またデータが実際にメモリに格納される順番にも気をつけます。ワード・データは上位下位が逆になって格納されます。



#### ＜リスト5＞ DEFB, DEFW の使用例

```

DEFB 10 ..... 1 バイトの10(16進数で0Ah)
DEFW 10 ..... 1 ワードの10(16進数で000Ah)

DEFB 'HELLO' ..... 5 バイトデータ
DEFB "HELLO" ..... 5 バイトデータ

DEFB 10, 20 ..... 二つ以上の定義はカンマで区切る
DEFB 'Hello', 0 ..... 文字列と数値を1行に書ける
DEFW MAIN ..... ラベルの値も入る

```

#### ＜リスト6＞ DEFS の使用例

```

ORG 8000H ;RAM領域

BUFF: DEFS 10 .....10バイトのメモリを確保
      ;メモリ領域の先頭アドレスは
      ;ラベルBUFF

```

このように、プログラム中で使用するパラメータや定数、変数の初期データを定義するものです。よってマイコンのプログラムでは、この疑似命令はROM領域で使用します。RAMは電源OFFでその内容は消えてしまいます。したがってRAM領域に初期データを定義することはできません。

しかし変数は、プログラム実行中に値を変更するわけですから、値を書き換えられるRAM上に割り当てられなければなりません。これには次のDEFSを使います。

#### ● DEFS

DEFSは、指定したバイト数のメモリ領域を確保します。例えばリスト6のようにすると、10バイトのデータ領域が確保され、その先頭アドレスはラベル名BUFFという名前になります。

これも省略形としてDSが使えるアセンブラもあります。

これは先述のDEFBやDEFWと違い、特定のデータを初期データとしてもたせることはできません。確保したメモリ領域の先頭アドレスと、それ以降に指定されたバイト数分のメモリ領域が確保されたにすぎません。

この領域をとくに初期化する必要がある場合は、その領域を使用する前に、プログラムで初期化する必要があります。

DEFSは、文字列の受信バッファとして使ったり、読み込みデータの一時退避バッファとして使うのが一般的なので、一般的にはRAM領域に定義して使います。

#### ● INCLUDE

INCLUDEは、指定されたファイルをそのソース・リストに取り込む命令です。アセンブラは

#### ＜リスト7＞ INCLUDE の使用例

```

INCLUDE "Z84C015.DAT"

OUT (PIOAD),A
OUT (PIOBD),A

```

(a) TEST.ASMの例

```

PIOAD EQU 1CH
PIOBD EQU 1EH

```

(b) Z84C015.DATの内容

INCLUDE文があった場合、そのソース・リストの位置に指定されたファイルを挿入し、一つのソース・ファイルであるとみなしてアセンブルします。

このINCLUDE疑似命令は、つぎのような場合によく使用されます。

- いくつものプログラム・ファイルがあり、それらに共通で利用されるEQU定義などをINCLUDEで取り込む
- とくに汎用的なプログラム・ファイル(ライブラリ)を取り込む
- 仕様の異なる部分だけをINCLUDEで取り込み、メイン部分は共通化する

リスト7に使用例を示します。TEST.ASMにはシンボルPIOADを定義している部分はありません。しかしZ84C015.DATをINCLUDEすることで、その部分にリスト(b)の文字列が埋め込まれたソース・リストであるとみなされるので、エラーは出ないで正常にアセンブルされます。

#### ● IF～ELSE～ENDIF

条件付きアセンブル命令で、条件式が成立したときはIF～ELSEの間をアセンブルします。成立しないときはELSE～ENDIFの間をアセンブルします。

リスト8にIF～ELSE～ENDIFの使用例を示します。このリストでは動作させるCPUがTMPZ84C015の場合は、シンボルCPUの値を015に、TMPZ84C011のときは011にします。

Z84C015の平行ポートの初期設定方法と、Z84C011の平行ポートの初期設定方法が異なるため、シンボルCPUの値によりCPUを判別して条件付きアセンブルをしている例です。

データを出力する方法はそのままOUT命令を実行すればよいので、ENDIFの後に記述します。ただしそのI/OアドレスはCPUによって異なります。このI/Oアドレスの定義も、条件付きアセンブルの中で定義されているので、どちらのCPUでも動作するソー

## <リスト8>

### IF~ELSE~ENDIF の 使用例

CPU	EQU	015	
IF CPU	EQU	015	
PIOAD	EQU	1CH	:PIO Ach データ・ポート
PIOAC	EQU	1DH	:PIO Ach コントロール・ポート
	LD	A, 11001111B	:モード3 ビット・モード
	OUT	(PIOAC), A	:全ビット出力
	LD	A, 11111111B	
	OUT	(PIOAC), A	
	LD	A, 00000111B	:割り込み使用しない
ELSE			
PIOAD	EQU	50H	:PIO Ach データ・ポート
PIOAC	EQU	54H	:PIO Ach コントロール・ポート
	LD	A, 11111111B	:全ビット出力
	OUT	(PIOAC), A	
ENDIF			
	LD	A, B	
	OUT	(PIOAD), A	:データ出力

TMPZ84C015のとき、  
この部分がアセンブル  
される

TMPZ84C011のとき、  
この部分がアセンブル  
される

ス・リストを記述できます。

ただしこの命令も、乱用すると見通しの悪い、読み  
にくいプログラムになります。

### ● CSEG, DSEG

CSEG はコード・セグメント、DSEG はデータ・セグ  
メントの宣言文です。コード・セグメントとは ROM  
(プログラムや定数データ)領域、データ・セグメント  
は RAM(変数データなど)領域と考えてください。

これらの疑似命令はリロケータブル型のアセンブラ  
に用意されているもので、効率的なモジュール化プロ  
グラミングを支援するものです。

CSEG/DSEG を各モジュールごとに宣言し、プロ  
グラムとデータ領域を明確に記述することで、最終的

なアドレス割り当てはプログラムを連結するリンクが  
決定し、プログラム領域やデータ領域を特に意識する  
必要はなくなります。

これらの疑似命令を使わないと、一つのメイン・モ  
ジュールですべての変数領域の割り当てを管理しなけ  
ればならなくなります。これではモジュール化する意  
味が損なわれてしまいます。

アプソリュート型アセンブラでは、とくに意味はあ  
りません。逆にアセンブラによっては対応していない  
命令としてエラーを出す場合もあります。

### ● 算術演算子

+, -, /, \*などは、アセンブラがもっている算  
術演算子です。まちがっても Z80 にこれらの算術演

## C 言語とアセンブラの上手な使い分け

筆者は5~6年前から、Z80 でも C コンパイラを  
中心として必要な部分だけアセンブラで ROM 化す  
る手法を用いています。

また、人間がアセンブラで作成したコードと、同  
様の処理を C で記述しコンパイルしたコードとを  
比較した場合、やはりコンパイラの出力するコード  
はまだまだ高速化の余地があります。

またサイズのにも、コンパクトになるようなコード  
生成を意識しないでプログラムした場合は2~3  
割、意識しても1割程度、プログラム・サイズが大き  
くなります。

しかし、このようなデメリットよりも、開発とデ  
バッグの効率が飛躍的に向上するメリットのほうが  
大きいので、ほとんど C で書くようにしています。

特にサブルーチンをライブラリ化するとき、アセ  
ンブラの場合にはどうしてもレジスタがこれこれ

…などといったことを考えなければなりませんし、  
ソース・プログラムの行数も多くなって読みにくく  
なってしまいます。

C ならばサブルーチンなど完全にブラック・ボッ  
クス化できるので、よけいなことに気をつかう必要  
はありません。

また、より高速化するためにはアセンブラで書い  
たモジュールを C のサブルーチン(関数)にするこ  
とで、問題はほとんどなくなります。

C のソースレベル・デバッガは、本格的に効率良  
く開発を進めたい方には必須でしょう。筆者は iD-  
1600A(コンピュテックス)という ROM エミュレー  
タ型のデバッガを使用しています。通常は単なる  
ROM エミュレータとして使っているだけですが、  
原因が突き止めにくいバグにはソース・レベルのス  
テップ実行でテストしています。



〈表3〉  
演算子のいろいろ

算術演算子	( ) + - / * MOD(剰余)
	SHR(右シフト) SHL(左シフト) AND OR XOR
論理演算子	EQ(等しい) NE(等しくない) GE(以上)
	LT(より小さい) GT(より大きい) LE(以下) NOT(ビット反転)
その他	HIGH(16ビットの値の上位8ビットを取り出す)
	LOW(16ビットの値の下位8ビットを取り出す)

注) 論理演算子はおもに IF 文で使用する。偽が“0”，真が“1”。

(a) 各種演算子の例

高位 ↑	( )
	単項の +, -
優先 順位	* / MOD SHL SHR
	2項の +, -
↓	LT LE EQ GE GT NE
	NOT
低位	AND
	OR XOR
	HIGH LOW

注) 優先順位が同位の場合は出てきた順番に処理される。

(b) 算術演算子の優先順位

〈リスト9〉 算術演算子の使用例

DATA	EQU	16
	LD	A, DATA*2
	OUT	(PIOAD), A
(a) 演算子を使ったリスト		
	LD	A, 32
	OUT	(PIOAD), A
(b) (a)と同じ動作のリスト		

〈リスト10〉 ロケーション・カウンタ(\$)の使用例

	JP	\$	;( 'JR \$' も同じ)
(a) 無限ループの例			
LOOP:	JP	LOOP	;( 'JR LOOP' も同じ)
(b) ラベルを使った無限ループの例			

算子が使えるわけではありません。一般的なアセンブラでは表3のような演算子が使用できます。

ほかにもいろいろな演算子が用意されているアセンブラもあります。また、ANDが .AND. や&などと、表現が若干異なる場合もあります。

ところで、アセンブラがもっている演算子とはどういう意味でしょう。リスト9をみてください。リスト9(a)もリスト9(b)も、アセンブラから出力される機械語データは同じ、つまりリスト9(a)の場合は計算結果が格納されるわけです。このようにアセンブラは、アセンブルと同時に計算もしてくれるわけです。

またすでにリスト8で示しているように、条件付きアセンブルの条件式にも使われます。

#### ● ロケーション・カウンタ(\$)

アセンブラが内部でもっている、現在アセンブル中のアドレスを示す変数で、命令コードやデータを格納するたびに、そのバイト数だけインクリメントされま

す。また ORG 命令による指定は、この変数に値を設定することと考えることもできます。

よく使われる例としてリスト10のような、無限ループにする例があげられます。説明するまでもないでしょうが、これはリスト10(b)とまったく同様な動作をします。

#### ● END

アセンブリ・ソース・ファイルが終わったことを宣言します。アセンブラは END 文をみつけると、それ以降の文を無視してアセンブル作業を終わります。

よってこれ以降のソース・ファイルの内容については、いっさい関知しません。END 文以降の行は、コメントと同じように使用することができます。

なおアセンブラによっては、END 文がなくてもファイルの最終に到達した時点で、アセンブル作業を終了するものもありますが、このようなアセンブラであっても END 文は入れるべきです。

〈野口智樹〉

(トランジスタ技術 1994 年 6 月号および 10 月号より)

## トランジスタ技術 SPECIAL No.48

### 特集 作れば解る CPU

ロジックICで実現するZ80とキャッシュ・マシン

B 5 判 160 頁 定価 1,723 円(税込)

本書は、コンピュータの構造や動作原理を理解するために、ゼロから出発して、オリジナル・コンピュータの設計から組み立て調整までの方法を解説した自習用のテキストです。

CQ出版社

好評  
発売中

## MS-DOS 用アプソリュート・ クロス・アセンブラの使い方

アセンブラにはアプソリュート型とリロケータブル型があることはすでに説明しました。リロケータブル型は大規模プログラムの開発に最適で、さらにマクロ命令と呼ばれる機能により命令を拡張できるなど、アセンブラでありながら高級言語並みに使えます。

ただしそのぶん機能がありすぎて、初心者の方には使いづらい面があるのも事実です。ここではいちばん基本的な機能をおさえた、アプソリュート型アセンブラ ZAA.EXE の使い方(入手方法については後述)を説明します。

### ● 特徴と動作環境、仕様

ZAA.EXE はインテル HEX 形式の実行形式ファイルやリスティング・ファイルを出力する、日本語 MS-DOS 用の Z80 アプソリュート・クロス・アセンブラです。動作環境や仕様を表 4 に、実行画面(ヘルプ・メッセージ表示時)のようすを図 8 に示します。

日本語 MS-DOS 用ですから、エラー・メッセージやヘルプ・メッセージなどが日本語で表示されます。

アプソリュート型としてひとつの機能は備わっています。前述の疑似命令もすべて使えます。算術演算子も使えます。

名前やラベルに使える文字は、A~Z, 0~9, “\_”, “.”, 漢字も使えます。ただし、先頭の文字に 0~9 の数字は使えません。また現在のロケーション・カウンタを示す値として \$ も使えます。

### ● アセンブル作業の流れ

まず普段使い慣れているテキスト・エディタでソース・リストを入力します。ファイルの拡張子は .ASM にします。

アセンブルは、DOS のプロンプトから次のように入力します。

>ZAA ファイル名 /H /S /L[リターン]

拡張子の .ASM は省略できます。ここで例えば Z80TEST.ASM というソース・リストをアセンブルすると、Z80TEST.HEX という HEX ファイルと、

<リスト 11> Z80.BAT の内容

```
ECHO OFF
REM
REM  ZAA用 アセンブル・バッチ・ファイル
REM
IF "%1" == "" GOTO HELP
ZAA %1 /H /S /L
GOTO QUIT

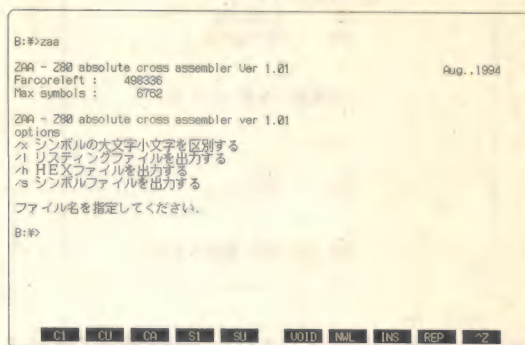
:HELP
ECHO アセンブルするファイル名を指定してください。

:QUIT
```

〈表 4〉 ZAA.EXE の動作環境および仕様や制限

動作環境	OS	日本語 MS-DOS Ver.3 以上
	メモリ	フリーエリア 384 K バイト以上
仕様や制限	<ul style="list-style-type: none"> <li>一行の長さは MAX130 文字程度</li> <li>シンボルは、長さ無制限(実際は数十文字程度)</li> <li>シンボル・ファイルで出力できるのは 44 文字まで</li> <li>ニーモニックはサイログ形式</li> <li>未定義命令(LD IXH, n)はサポートしない</li> <li>アセンブルした結果が 64 K を超えるようなことがあっても無視する</li> <li>シンボル数の最大値は、起動時のメモリの残りによって決定される</li> <li>EQU の前方参照は不可</li> </ul>	

〈図 8〉 ZAA の起動画面



Z80TEST.LST というリスティング・ファイル、Z80TEST.SYM というシンボル・ファイルを出力します。

インテル HEX とは、ROM ライタなどにプログラムを書き込むときなどによく使われるファイル形式で、拡張子が .HEX となります。後述する ROM エミュレータ型 ICE でも使用します。

リスティング・ファイルとは、ソース・リストにアセンブルした機械語やアドレス情報を付加したリストです。シンボル・ファイルとは、そのプログラムの中で使用されたラベルや名前などを示すファイルです。

なお、これらのファイル形式などの中身については特に知らなくてもとりあえずはかまいません。

アセンブルするたびにこのようなオプション・スイッチを付けるのも面倒なので、リスト 11 のようなバッチ・ファイルを作って使いやすくします。使い方は、>Z80 ファイル名[リターン]

とするだけです。

### ● エラーへの対処

アセンブル時に表示されるエラーメッセージは、シンボルやラベルが定義されていないときや、文法上のエラーなどがあったときに表示されます。エラーが発生すると何行目が何のエラーかを表示するので、先ほどソース・プログラムを入力したときと同様に、テキ



スト・エディタでソース・プログラムを呼び出して、エラーのあった行を修正します。

図9にエラーが発生したときの画面表示のようすを示します。

### ● エラー対処例

ここではいくつか代表的なエラーと、その修正方法や対策方法を説明します。

#### ▶ アセンブルできないコード

Z80では使えない命令を指定したときなどに表示されます。

```
ADD      C,B
```

CレジスタとBレジスタの直接の足し算はできません。Z80は命令の直交性が悪いので、ありそうでない命令を使ってしまうことがよくあります。命令表でよく確認してください。

#### ▶ 0で除算した

その名のとおりゼロで割り算したときに表示されます。もっとも普通は、

```
LD      A,5/0
```

などと直接ゼロを書くことはないと思われます。

```
LD      A,10/DATA
```

などのとき、DATAの値がゼロのときに表示されます。

#### ▶ 定数が不正

文字列を切り出して数値に変換するとき、文字が残った場合などに表示されます。

```
LD      HL,1A6
```

文字の最後に"H"がないと16進数とはみなされません。または途中の"A"が余計です。

```
LD      A,0002000B
```

2進数表記なのに"2"が入っています。

#### ▶ 値がない

本来なら値や文字がくるべき所になにもない则表示されます。

```
LD      (IX+),1
```

```
DEFB    1,2,
```

#### ▶ 括弧などの対応がおかしい

括弧やダブルコーテーションの囲みの対応がおかしいときに表示されます。

```
DEFB    "ABVC
```

```
INCLUDE "IOADR.H
```

最後にダブルコーテーションが足りません。

```
DEFB    'DKJLFKAJDL'
```

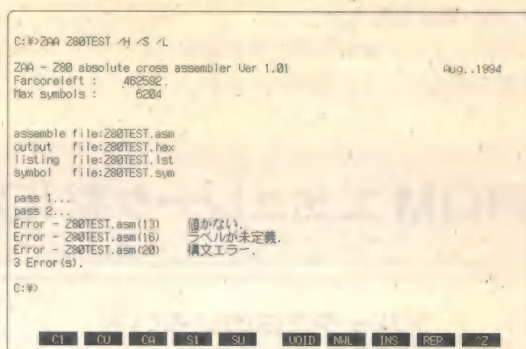
文字列の囲みはシングルコーテーションでもダブルコーテーションでもできますが、どちらかに統一してください。

```
LD      A,(DATA1+(DATA2+1)
```

計算式の括弧が足りません。

```
LD      (HL+1),A
```

〈図9〉ZAAのエラー・メッセージのようす



HLレジスタではディスプレイメント付きのインデックス・アドレッシングはできません。IXやIYレジスタを使えば大丈夫です。

#### ▶ 余分な文字列が付いている

命令や構文上、文字のいらないうちに文字がきていると表示されます。

```
LD      A,B :comment
```

コメントの指定に":"を間違って使った例です。

```
LD      A,(DATA+2))
```

括弧が多いときも表示されます。

#### ▶ ラベルまたは名前が不正

ラベルには予約語や先頭が数字の文字列は使えません。

```
LD      A,LD+4
```

#### ▶ ラベルまたは名前の二重定義

同じ名前のラベルや名前があるときに表示されます。

```
name EQU 100H
```

```
name: LD A,H
```

#### ▶ ラベルが未定義

定義されていないラベルを使うと表示されます。

```
LD      A,hjk
```

ラベルの綴り間違いということが多いため、スペルを確認してください。また、

```
LD      A,B
```

のように","と間違つて"."を使つてもこのエラーが表示されます。

#### ▶ 構文エラー

ラベル、名前、予約語以外の文字で行が始まるなどに表示されます。

```
100: LD A,B
```

先頭の文字が数字の場合はラベルになりません。

```
LD: XOR A
```

予約語はラベルに使えません。

```
(IX+4),A
```

いきなりオペランドがきている例です。

```
LD      B,
```

”、”の次がありません。

▶ 値が範囲を超えた

相対ジャンプ命令で飛び先が相対アドレッシングの範囲を超えたときに表示されます。

JR

Z,JP\_ADR1

JP 命令にするか、飛び先のアドレスに近い位置に変更します。

## ROM エミュレータ型 ICE を使ったデバッグのすすめ

### デバッグ方法のいろいろ

#### ● デバッガとは

すでに説明したように、アセンブラの表示するエラー・メッセージとは、文法上のエラーについてだけで、プログラムの論理的な間違いについては関知していません。

よって CPU がどのようにプログラムを実行していくか、レジスタやメモリはどう変化していくかを見ることができると、プログラムの動作そのものわかり、プログラムの動作のチェックが非常にしやすくなります。

このように、プログラムを実行したときのレジスタの変化のようすや、メモリの内容を表示したり編集できたりするツールを、デバッガとよびます。

#### ● ROM ライタだけ！

デバッグ・ツールが何もない状況では、できたプログラムを ROM に書き込んで動作するかどうかを確認する方法しかありません。動作しなければまたプログラムを直して再アSEMBルし、ROM の書き直しが必要です。

また、パソコンの画面やプリンタに打ち出したリストを眺めるという机上デバッグしか手がないので、タイミングに問題のある場合などは、その原因を突き止めるのが困難になります。

#### ● ターゲット側にモニタを乗せる

次に考えられる手法としては、動作させるマイコン

側に、何らかのモニタ・プログラムを搭載し、このモニタ・プログラムを使いながらレジスタやメモリの変化を確認しプログラムの動作を調べるものです。

スタンド・アロンで動作させたいなら、最低限 16 進キーによる入力や 7 セグメント LED による表示が必要です。しかしデバッグが終われば不要になります。組み込み機器としてはデバッグ時だけのハードウェアは無駄になります。このスタイルのマイコン・ボードは基本的に学習用ということになるでしょう [図 10 (a)]。

そこで、入力や表示は他の装置にまかせ、ターゲット側にはモニタ・プログラムとその装置との通信部分のハードウェアを搭載させる形のモニタが考えられました。これがリモート・モニタとよばれるものです [図 10 (b)]。

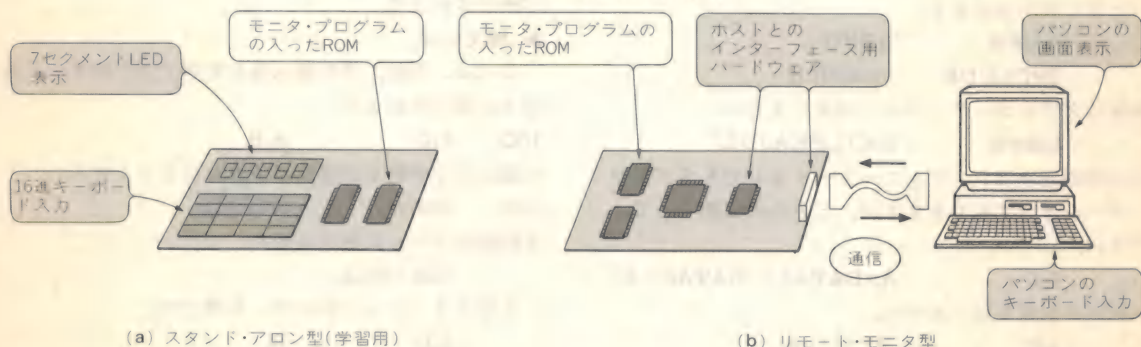
モニタ・プログラムの機能としては、レジスタやメモリの表示や書き換えの機能は同じですが、その結果をシリアル・ポートなどを通じてパソコンなどとやり取りする部分がスタンド・アロン型と異なります。

#### ● リモート・モニタを使ったデバッグ

ROM ライタだけの環境よりは格段に“まともなデバッグ”ができる環境にはなりました。しかし、ホスト側からデバッグしたプログラムを転送して実行したり、メモリを書き換えたりしてデバッグを進めるわけですから、RAM 上で作業しなければなりません。ROM 領域でのメモリの書き換えやステップ実行などが不可能です。

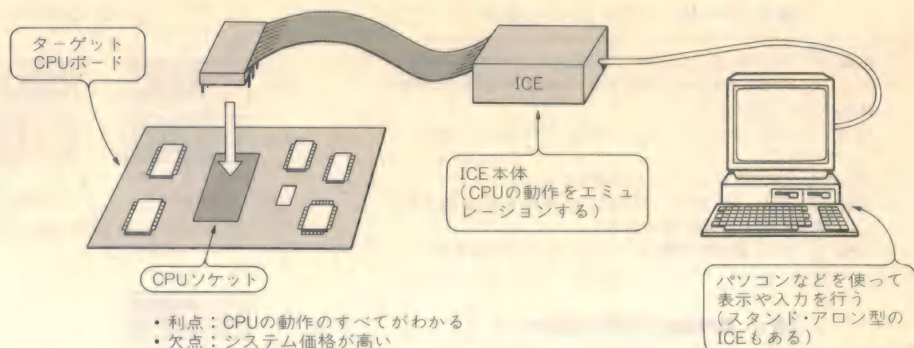
またマイコンの限られたメモリ空間に、モニタ・プ

〈図 10〉 モニタ・プログラムの形態

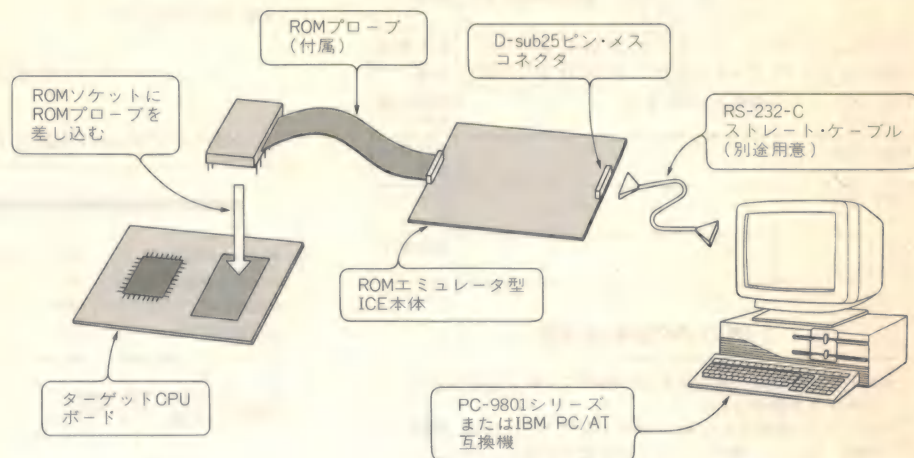




〈図 11〉  
ICE によるデバッグ



〈図 12〉  
ZVP のシステム構成



プログラムも入るわけですから、開発するプログラムが大規模になるとメモリが足りなくなることもあります。

### ● ICE によるデバッグ

ICE(イン・サーキット・エミュレータ)とは CPU そのものをエミュレーションさせることで、CPU の動作をその手の内までも調べられる、マイコン開発ツールとしては最高のデバッグ・ツールです(図 11)。

CPU に電源やクロックが正常に入力されているかどうかの物理的なチェックから、ROM 領域のメモリの読み書き、ステップ実行なども可能になります。また基本的にマイコンのメモリ空間にモニタ・プログラムのための領域やワークエリアが取られることはありません。

しかしやはりネックはその価格にあるでしょう。Z80 用といえど、多機能なものは数十万円はします。また使用する CPU のパッケージが QFP だったりすると、取り付けのためのアダプタが必要になったり、取り付け不能場合すらあります。

### ● ROM エミュレータ型 ICE によるデバッグ

ここでは、ICE と同等の機能を持ちながら、それでいてリモート・モニタ並の価格と使いやすさの、ROM エミュレータ型 ICE Z Vision PRO(システムロー

ド)を活用したデバッグ手法について簡単に解説します。

## Z Vision PROmini

### ● ZVP の概要

Z Vision PRO(以下 ZVP)とは、ROM プローブをターゲットの ROM ソケットに、RS-232-C 端子をホスト・パソコンに接続して使用する、ROM エミュレータ型 ICE です(図 12)。

ここでは ZVP の機能限定版である Z Vision PROmini を使用してみます。mini 版といっても、ひと通りのデバッグ機能は備わっています。

その主な特徴は、

- ▶ ROM 領域へのダウンロード、ブレーク・ポイントの設定、ステップ実行が可能
  - ▶ Z80 のもつメモリや I/O の全アドレス空間をユーザに解放
  - ▶ ROM ソケットに接続するので、取り付け方法が CPU のパッケージに左右されない
- などが上げられます。

対応するターゲット CPU ボードの条件を表 5 に、

〈表 5〉 ターゲット CPU ボードの条件

- ・搭載 CPU が Z80 または Z80 とオブジェクト互換の CPU。
- ・アドレス 0000h 番地から最低 8 K バイトの空間が ROM 領域になっている。
- ・64 K、128 K、256 K ビットの ROM に対応した ROM ソケットであること。
- ・RAM を最低 2 K バイト以上装備している。
- ・RESET 信号、 $\overline{M1}$  信号、MREQ と WR を負論理 AND した MWR 信号が取り出せること。
- ・CPU のバスと ROM の間にバッファなどがないもの。

〈表 6〉 動作確認済み動作保証ボード

・ボード型名	最高クロック周波数	メーカー
AK180	最高クロック周波数 8 MHz (ただし、キットなので完全に動作が確認されているもののみ)	秋月電子通商
A3000 シリーズ A5000 シリーズ	ウェイトありで、10 MHz まで対応 ウェイトありで、16 MHz まで対応	亜土電子工業
UEC-Z77 シリーズ	最高 8 MHz まで	梅澤無線電機
KBC-Z50		共立電子産業
GPY-Z15		日本コムネット
TC-11 TC-15		田中電子

〈表 7〉 ZVP 使用上の制限

- ・アドレス 0000h~007Fh までの領域には割り込みベクタ・テーブルを置かない
- ・モニタに復帰するための命令として RST 00H, 08H, 30H, 38H の 4 本のうちの 1 本を使用する
- ・CPU の最高クロック周波数は 8 MHz まで (ウェイト挿入ができれば 16 MHz まで)

動作確認済みのマイコン・ボードを表 6 に示します。

またメモリや I/O の制限は基本的にはありませんが、表 7 に示すような若干の制限があります。

#### ● 前準備

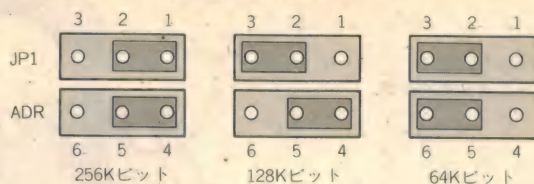
まず使用するターゲット・ボードに合わせて各モード・スイッチをセットします。まずターゲット ROM ソケットに合わせて ROM 容量の切り替えをします。さらに CPU についても図 13 のように切り替えます。それ以外のジャンパ・スイッチは標準状態でかまいません。

次にターゲット・ボードの ROM ソケットへ ICE の ROM プローブを挿入します。ピン方向を間違えないでください。

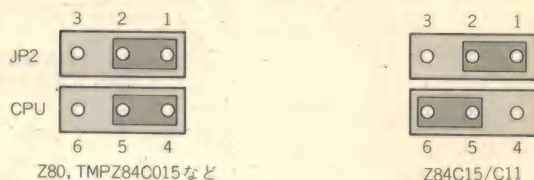
そして CPU の RESET 端子に RST を、 $\overline{M1}$  端子に  $\overline{M1}$  を、SRAM の  $\overline{WE}$  端子に  $\overline{WE}$  の各 IC クリップを接続します。CPU やメモリがフラット・パッケージなどを使っている、IC クリップの接続ができないときは、その信号が接続されているコネクタや拡張端子などでかまいません。

次にホスト・パソコンと ICE を RS-232-C のストレ

〈図 13〉 ZVP のジャンパ設定

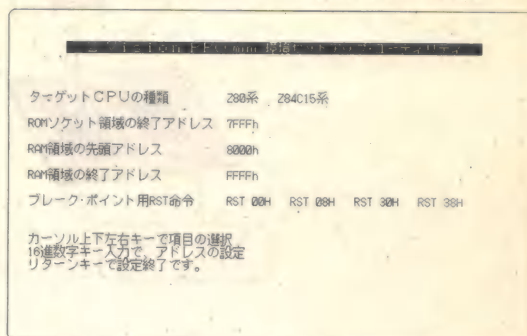


(a) ROM容量の設定 (JP1)



(b) CPUの設定 (JP2)

〈図 14〉 Z Vision PROmini のセットアップ画面



ート・ケーブルで接続します。ホスト・パソコン側のシリアル端子は、パソコンの機種やデスクトップ型やノート型などの違いでいろいろあるので、ケーブルは各自使うパソコン用のストレート・ケーブルを用意してください。ICE 側は D-Sub25 ピン・メスです。

そして接続間違いがないかどうかよく確かめてから、ターゲット・ボードの電源を入れます。

#### ● ホスト・パソコン側の準備

まず MS-DOS を起動したら、添付フロッピー・ディスクに入っている環境セットアップ・プログラム SETZVP を実行します。

>SETZVP [リターン]

起動画面を図 14 に示します。ここでは使用するマイコン・ボードの CPU の種類、メモリ・マップ、プログラマブル・ウェイト・ステート・ジェネレータ内蔵の CPU のときは、そのウェイト数などを設定します。

設定が完了したら終了してください。

これ以降、同じマイコン・ボードを使う場合は再設定は必要ありません。メモリ・マップを変えたり、別のマイコン・ボードを使用するときにもう一度設定し



てください。

そしてZVPを起動します。起動方法は、  
>ZVPM[リターン]

とするだけです。オプション・スイッチは特に必要ありません。

### ● ZVP の画面構成

図 15 に起動時の画面のようすを示します。

①をダイアログ・ウィンドウと呼び、コマンドの入力はここで行います。②をディスプレイ・ウィンドウと呼び、逆アセンブル・リストが表示されます。③をレジスタ・ウィンドウと呼び、各レジスタの値やフラグのようすを示します。④をダンプ・ウィンドウと呼び、メモリの16進ダンプ表示をします。④はダンプ・ウィンドウのASCII文字表示画面です。

⑤はブレーク・ウィンドウで、ブレーク・ポイントが表示されます。⑥はZVPの動作モードの表示で、ターゲット・ボードと通信しているときは通信中、Gコマンドでプログラム実行中は実行中、コマンド待ちの状態のときは休止中と表示します。

## ZVP の使い方

### ● プログラムのロード

まず何はともあれ、デバッグしたいプログラムをロードしなければなりません。ZVPでロードできるプログラム・ファイル形式は、インテルHEXファイル形式です。またファイル名が同じで、拡張子が.SYMというシンボル・ファイルがあると、そのファイルも読み込みます。

カーソルがダイアログ・ウィンドウにいるときに、F2を押すとロード・ファイル形式の選択画面になります。ここでカーソルをHEX形式に移動しリターン・キーを押すと、HEXファイルの一覧が表示されます。カーソル・キーで読み込みたいファイルを選択しリターン・キーを押せば読み込みます。

またLコマンドから読み込むときは、ダイアログ・ウィンドウで、

ZVP>L ファイル名 ファイル名 [リターン]  
としても読み込みめます。

### ● ダンプ・ウィンドウの表示/編集

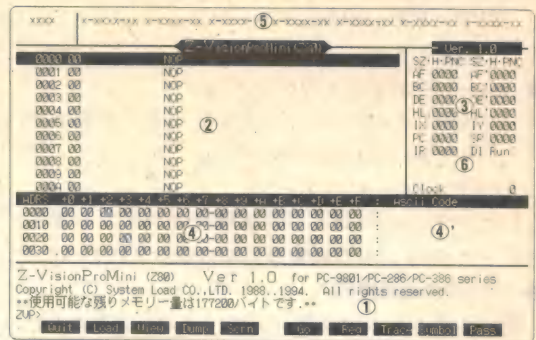
ダンプ・ウィンドウに任意のアドレスのデータを表示させたいときは、ダイアログ・ウィンドウでDコマンドを使います。

ZVP>D 8000[リターン]

これでアドレス8000h以降のメモリが16進ダンプ画面で表示されます。

内容を編集したいときは、ダイアログ・ウィンドウでF4キーを押すとカーソルがダンプ・ウィンドウに移動します。ここで数値を入力すればOKです。また

〈図 15〉 Z Vision PROmini の起動画面



カーソル・キーで移動すると、表示範囲がスクロールします。またもう一度F4キーを押すとダイアログ・ウィンドウにカーソルが戻ります。

### ● ディスプレイ・ウィンドウの表示/編集

ディスプレイ・ウィンドウに任意のアドレスのリストを表示させたいときは、ダイアログ・ウィンドウでUコマンドを使います。

ZVP>U 100[リターン]

これでアドレス0100h以降の命令がアセンブル・リストで表示されます。またダイアログ・ウィンドウでF5キーを押すと、カーソルがディスプレイ・ウィンドウに移動し、カーソル・キーで上下方向に移動するとスクロールします。もう一度F5キーを押すとダイアログ・ウィンドウにカーソルが戻ります。

命令を書き換えたいときは、ダイアログ・ウィンドウから、

ZVP>A 200[リターン]

とすると、アドレス200h番地から、1パス・アセンブルができます。1パス・アセンブルとは、1行入力するごとに機械語に変換してくれる機能です。何も入力しないでリターン・キーを押すと、ダイアログ・ウィンドウに戻ります。

### ● レジスタの値の変更

レジスタの値を変更するときは、ダイアログ・ウィンドウからF7キーを押します。するとカーソルがレジスタ・ウィンドウに移動し、カーソル・キーで変更したいレジスタに移動できます。そして数値を入力すると値を変更できます。リターン・キーを押すか、もう一度F7キーを押すとダイアログ・ウィンドウに戻ります。

またダイアログ・ウィンドウから直接、

ZVP>R PC 8000[リターン]

と、レジスタ名と値を指定しても設定できます。

### ● ブレーク・ポイントの設定

ブレーク・ポイントとは、実行を止めたいアドレスのことで、

ダイアログ・ウィンドウから、  
ZVP>BP 29A1[リターン]  
として設定します。

### ● ブレーク・ポイントのクリア

一度設定したブレーク・ポイントをクリアします。  
ダイアログ・ウィンドウから、  
ZVP>BC n[リターン]  
として、ブレーク・ポイント番号 n をクリアします。  
ブレーク・ポイントは画面最上部に表示されています。

### ● プログラムの実行

実行は G コマンドを使います。プロンプトから、  
ZVP>GO[リターン]

とすると、アドレス 0000h から実行を開始します。アドレスがブレーク・ポイントになったら実行をやめてモニタに戻ってきます。

また直接ブレーク・アドレスを指定することもできます。

ZVP>GO 290[リターン]

この例では、アドレス 0000h から実行を始めて、アドレスが 290h になるとブレークしてモニタに戻ってきます。

### ● ステップ実行

ステップ実行とは、命令を 1 ステップずつ実行していくものです。1 命令の実行が終わるたびに画面の各

〈表 8〉 Z Vision PROmini のコマンド一覧

コマンド	内 容	書 式
A	1 バス・アセンブルを行う	A[<アドレス>]
BP	ブレーク・ポイントの設定	BP[<n>][<アドレス>][<回数>]
BD	ブレーク・ポイントの有効化	BD n <sub>0</sub> [n <sub>1</sub> [n <sub>2</sub> …]]
BE	ブレーク・ポイントの有効化	BE n <sub>0</sub> [n <sub>1</sub> [n <sub>2</sub> …]]
BC	ブレーク・ポイントの削除	BC n <sub>0</sub> [n <sub>1</sub> [n <sub>2</sub> …]]
BL	ブレーク・ポイントのリスト	BL
BI	ブレーク・ポイントの通過回数の再設定	BI n <sub>0</sub> [n <sub>1</sub> [n <sub>2</sub> …]]
D	メモリ・ダンプの表示と書き換え	D[<モード>][<開始アドレス>][<終了アドレス>]
F	メモリをデータで埋める	F<開始アドレス><終了アドレス>d <sub>1</sub> [d <sub>2</sub> …]
G	プログラムの実行	G[<開始アドレス>][<ブレーク・ポイント>]
ID	指定ポートからデータを読み込む	ID[I/O アドレス]
L	インテル HEX ファイルの読み込み	L[<HEX ファイル>][<シンボル・ファイル>]
MV	メモリのコピー	MV<開始アドレス><終了アドレス><転送先アドレス>
OD	指定ポートへのデータの書き込み	OD[I/O ポート]<出力データ>
P	バス・トレース	P[<回数>]
Q	終了	Q
R	レジスタ、フラグの設定	R[<reg>][<値>]or RF<flgs>[<flgs>[<flgs>…]]
SL	シンボルの表示	SL[<シンボル>]
SC	シンボルの削除	SC<シンボル>
SS	シンボルの登録	SS<シンボル><値>
T	ステップ・トレース	T[<回数>]
U	逆アセンブル・リストの表示	U[<アドレス>]
WH	インテル HEX ファイルの出力	WH[<ファイル>][<開始アドレス>][<終了アドレス>]]

注) ダイアログ・ウィンドウのコマンドと指定形式。[] パラメータは省略可。  
<>でくくられたパラメータは 0~0FFFFh(16 進)の指定が可能。

(a) コマンド一覧

10 進数	先頭が%で始まる場合	%124(10 進数の 124)
16 進数	先頭が 0~9 で始まる場合 A~F の文字で始まる場合は先頭に 0 を付ける	8000(10 進数の 32768) 0FFFFh(10 進数の -1)
シンボル	先頭が数字以外の文字(ただし小文字は大文字に変換される) シンボルに使用できる文字 A~Z 0~9 \$ . ? @ _	BC("BC"というラベル名)
レジスタ	先頭が#で始まり、以下のレジスタ名文字列がついたとき A,B,C,D,E,H,L,I,R,BC,DE,HL,IX,IY,PC,SP A',B',D',E',H',L',BC',DE',HL'	# BC(BC レジスタに格納されている値)
演算子	+のみ使用可能	8000+10(16 進数で 8010h)

(b) 定数、数値の表現



ウィンドウに直後の状態が表示されるので、レジスタの変化のようすが手に取るようになります。

まず R コマンドなどで実行開始アドレスを PC レジスタに設定します。そしてダイアログ・ウィンドウで F8 キーを押すと、PC レジスタで示されるアドレスの命令を 1 命令実行して戻ります。各レジスタやメモリ内容は、その実行した命令によって変化します。当然 PC レジスタも次の命令のアドレスを示します。

ですから一度押すと次の命令を、さらに押すとまた次の命令をとという具合に、1 命令ずつ実行を繰り返します。

### ● 連続ステップ実行

先ほどのステップ実行と同様に PC レジスタに実行開始アドレスを設定してから、ダイアログ・ウィンドウで SHIFT キーを押しながら F8 キーを押します。ESC キーを押すまで連続的にステップ実行します。

### ● パス・トレース実行

ステップでは、サブルーチンがコールされても、サブルーチンも 1 ステップずつ実行していました。パス・トレースとはサブルーチンなどのコールがあったときは、そのサブルーチン内はブレイクせずにリアルタイムで処理するトレースです。

サブルーチン・コールをそれ自体があたかも 1 命令であるかのようにステップ実行できるので、メイン・ルーチンの実行の流れをチェックしたいときに使用します。これもステップ実行と同様に PC レジスタにア

ドレスを設定したら、ダイアログ・ウィンドウで F10 キーを押します。

### ● 連続パス・トレースの実行

パス・トレースを連続で行うモードです。使い方は連続ステップと同様です。ESC キーで終了します。

### ● 終了

ダイアログ・ウィンドウで、F1 キーまたは、

ZVP>Q[リターン]

とすると、終了の確認を聞いてくるので、Y を押せば終了します。

表 8 に ZVPmini 版のコマンドの一覧を示します。

### ● デバグの手順

まずデバグするプログラムをロードします。ROM 領域に対しても読み書きできるので、プログラムがアドレス 0000h から始まるものでもかまいません。

デバグのコツはブレイク・ポイントの設定とトレースの使い方にあります。正しく実行することがわかっていない部分をトレースしても時間の無駄です。そこでここから先の動作を調べたいというアドレスにブレイク・ポイントを設定し、実行してブレイクさせます。そこからステップ実行やトレースをさせて、レジスタやメモリの変化を調べるというやり方が一般的です。

〈藤丘勝信〉

### ◆参考文献◆

- (1) Z Vision PRO ユーザーズマニュアル, (株)システムロード。

## 頒布ディスクの内容と申し込み方法

### ● 頒布ディスクについて

本特集に掲載したリストをディスク頒布します。

ディスクの内容は、第 1 章から第 11 章まででできたすべてのリストと、Z80 用アプソリュート・クロス・アセンブラ ZAA です。

ZVP については目次裏の申し込み方法をご覧ください。

### ● 頒布ディスクの申し込み方法

下記申し込み用紙に必要事項を記入し、代金同封のうえ現金書留で、右記宛先までお送りください。

▶ 頒布価格 3,000 円(税, 送料込み)

#### ▶ 頒布メディア

3.5 インチ 2HD(1.25M フォーマット)

3.5 インチ 2HD(1.44M フォーマット)

5 インチ 2HC(PC9801 読み込み可)

#### ▶ 申し込み期限

1998 年 12 月 31 日(当日消印有効)

#### ▶ 申し込み先

〒170 東京都豊島区巣鴨 1-14-2 CQ 出版(株)

トランジスタ技術 SPECIAL No.49

ディスク頒布係

## ● トランジスタ技術 SPECIAL No.49 ソース・リスト申し込み用紙

TRSP49Z

送り先ご住所: 〒

お名前:

〒 ( ) -

▶ 希望メディア(✓印を付けてください)

☐ 3.5 インチ 2HD(1.25M)

☐ 3.5 インチ 2HD(1.44M)

☐ 5 インチ 2HC



- 本書掲載記事の利用についてのご注意 — 本書掲載記事には著作権があり、また工業所有権が確立されている場合があります。したがって、個人で利用される場合以外は所有者の承諾が必要です。また、掲載された回路、技術、プログラムを利用して生じたトラブル等については、小社ならびに著作権者は責任を負いかねますのでご了承ください。
- 本書に関するご質問について — 文章、数式等の記述上で不明な点についてのご質問は、必ず往復はがきか返信用封筒を同封した封書にてお願いいたします。ご質問は著者に回送し直接回答していただきますので、多少時間がかかります。また、本書の範囲を超えるご質問には応じられませんので、ご了承ください。

# トランスタ技術 SPECIAL

No.49

©CQ出版(株) 1995

編集人 蒲生良治  
発行人

発行所 CQ出版株式会社 ☎170 東京都豊島区巢鴨1-14-2

電話 03-5395-2121(出版部), 03-5395-2141(販売部)

振替 00100-7-10665

1995年1月1日 初版発行  
1997年7月20日 第3版発行

(定価は表四に表示してあります)

印刷・製本 三晃印刷株式会社



3端子/チョッパ/フライバック各種レギュレータ ICの使い方

トランジスタ技術編集部 編

## 電源用 IC活用マニュアル

B5判 160頁

定価1,682円

本書は各種電源用ICのデータの要点と活用方法を網羅しています。

各メーカーのデバイスの中から汎用の電源ICを精選し、タイプ別に電源の設計法や回路例、実測データなどポイントをおさえてわかりやすく解説をしています。

いまや電源システムは、エレクトロニクスの世界では必須であり、その心臓部とも言える役割りをこのICが担っています。技術進歩に伴い、電源ICも高効率で優れたものが開発され、さらに用途別に多種多様の製品が世に出まわっています。

電源システムを構築するための必携マニュアルとして本書をお勧めします。

抵抗、コンデンサ、インダクタ、機構部品の特徴と仕様

薊 利明/竹田俊夫 著

## わかる電子部品の基礎と活用法

B5判 184頁

定価1,733円

本書では抵抗、コンデンサ、インダクタ、機構部品の種類とその構造、仕様、特徴をイラストを豊富に使ってわかりやすく解説しています。それに加え、部品の故障率や故障モードなど高信頼設計のための基礎データなどもまとめてみました。ハードウェア・エンジニアには必読の書です。

計測制御の信号処理からセンサ/通信インターフェースまで

トランジスタ技術編集部 編

## モジュール化に役立つ実用電子回路集

B5判 160頁

定価1,631円

本書では、あらゆる場面で役立つ、モジュール設計のための回路として、汎用部品でコンパクトに構成した粋な回路を集めました。また設計した回路をより実用的なものにするために、モジュール化設計した回路同士やパソコン、測定器との接続などに役立つ、便利なインターフェース回路も豊富に紹介しています。

DOS/Vマシンのインターフェースを拡張するハードウェア設計 トランジスタ技術編集部 編

## IBM PCとISAバスの活用法

B5判 164頁

定価1,835円

本書ではIBM PC/AT互換機の標準入出力インターフェースの仕様をまとめたあと、ISAバスのハードウェアについて詳細に解説しています。さらに、16550Aを使用した拡張シリアル・ポート・アダプタ、高速FIFOを使用したファンクション・ジェネレータ・ボードなど、IBM PC/AT互換機用のISA拡張アダプタ・カードの設計・製作事例を具体的な回路図とサンプル・プログラムを示しながら解説しています。

### 新つくるシリーズ

●B5判●160頁●各定価1,529円●

#### No.1 〈好評発売中〉 つくるツール&測定器

おもな内容●デジタル電圧計／ファンクション・ジェネレータ／カーブ・トレーサ／LCメータ／etc.

#### No.2 〈好評発売中〉 つくるオーディオ&ビデオ

おもな内容●オーディオ・アンプ／サウンド・プロセッサ／ビデオ・セクタ／ビデオ・エフェクタ／etc.

#### No.3 〈好評発売中〉 つくるオリジナル・グッズ

おもな内容●電子ゲーム／キッチン・タイマ／電子温度計／電磁波時計／ニカド電池充電器／紫外線メータ／etc.



ISBN4-7898-3241-4

C3055 ¥1641E

**CQ出版社**

定価：本体1,641円（税別）



9784789832410



1923055016415

